	Type	L #	Hits	Search Text	DBs	Time Stamp
1	BRS	L2	2212	host adj2 bridge	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:13
2	BRS	L1	7	cpu adj task adj priorit$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:08
3	BRS	L3	1077	task adj priorit$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:11
4	BRS	L4	9	2 same 3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:08
5	BRS	L5	2716	task near priorit$3	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:11
6	BRS	L6	9	2 same 5	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:12

	Type	L #	Hits	Search Text	DBs	Time Stamp
7	BRS	L7	0	6 not 4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:12
8	BRS	L8	375260	bridge	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:13
9	BRS	L9	28	3 same 8	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:13
10	BRS	L10	19	9 not 4	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:28
11	BRS	L11	10	("5283904" "5758169" "5848279" "5857090" "5511200").pn.	USPAT; US-PGPUB; EPO; JPO; DERWENT; IBM_TDB	2004/04/30 11:29

US005564060A

United States Patent [19]

Mahalingaiah et al.

[11] **Patent Number: 5,564,060**

[45] **Date of Patent: Oct. 8, 1996**

[54] **INTERRUPT HANDLING MECHANISM TO PREVENT SPURIOUS INTERRUPTS IN A SYMMETRICAL MULTIPROCESSING SYSTEM**

[75] Inventors: **Rupaka Mahalingaiah; Rodney Schmidt**, both of Austin, Tex.

[73] Assignee: **Advanced Micro Devices**, Sunnyvale, Calif.

[21] Appl. No.: **251,849**

[22] Filed: **May 31, 1994**

[51] **Int. Cl.**[6] **G06F 9/00**

[52] **U.S. Cl.** **395/871**; 395/733; 395/550; 395/737; 364/DIG. 1; 364/271.6; 364/271.7; 364/DIG. 2; 364/950; 364/950.2

[58] **Field of Search** 395/500, 375, 395/741, 733, 737, 871, 868, 800, 550

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,237,674	4/1987	Mohme et al.	395/425
5,367,689	11/1994	Mayer et al.	395/725
5,446,910	8/1995	Kennedy et al.	395/800
5,455,916	10/1995	Bourke et al.	395/285
5,495,615	2/1996	Nizar et al.	395/733

OTHER PUBLICATIONS

Intel, "MultiProcessor Specification", Version 1.1 (Apr. 1994).

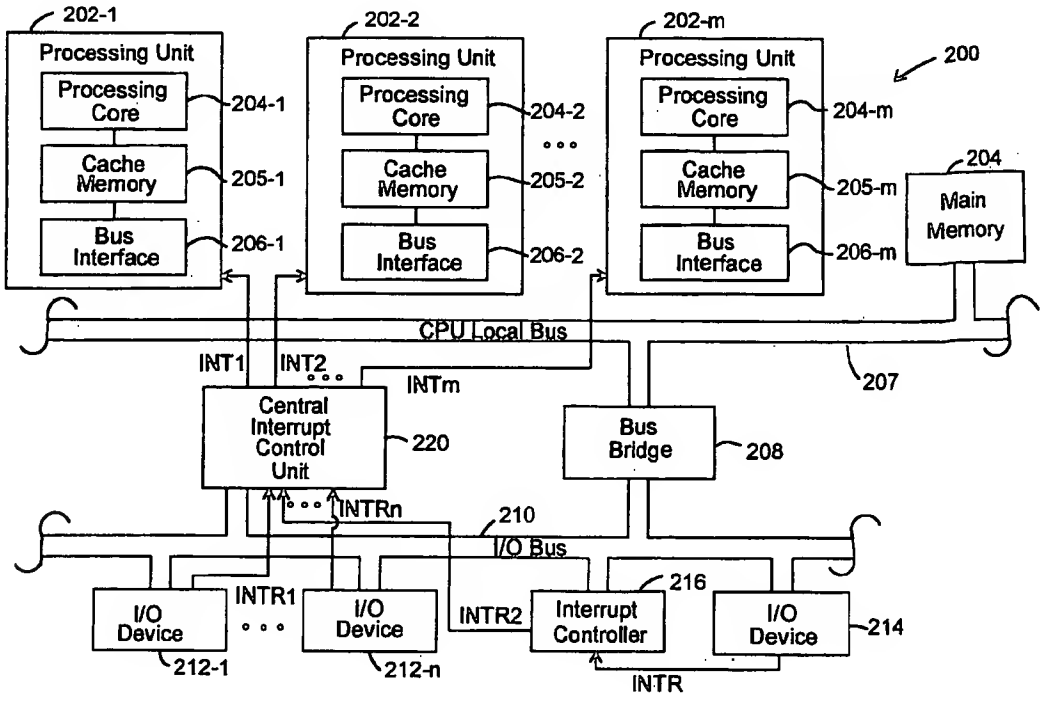
Primary Examiner—Larry D. Donaghue

Attorney, Agent, or Firm—B. Noel Kivlin

[57] **ABSTRACT**

A symmetrical multiprocessing system is provided that includes centralized interrupt control unit. The interrupt control unit is coupled to a plurality of processing units and to a plurality of interrupt sources. The interrupt control unit advantageously allows for the expansion of each interrupt pin by setting the interrupt control unit in a cascade mode. Furthermore, the central control unit is responsive to specialized interrupt cycles which allows I/O devices and/or bus bridge devices to initiate of an interrupt without requiring a dedicated interrupt line. The central interrupt control unit further allows each interrupt to be prioritized independently of its associated vector ID, and prevents the occurrence of spurious interrupts by providing a programmable latency timer which causes the central interrupt control unit to delay its response to End Of Interrupt (EOI) instructions. An auto-chaining technique is further implemented by the central interrupt control unit to sequentially provide broadcast interrupts to various processing units based on their current task priority values. Finally, the central interrupt control unit further handles system management interrupts (SMIs) from sources such as power management units and ensures proper system operation even if the requested system management function affects operations being carried by other processing units.

7 Claims, 22 Drawing Sheets



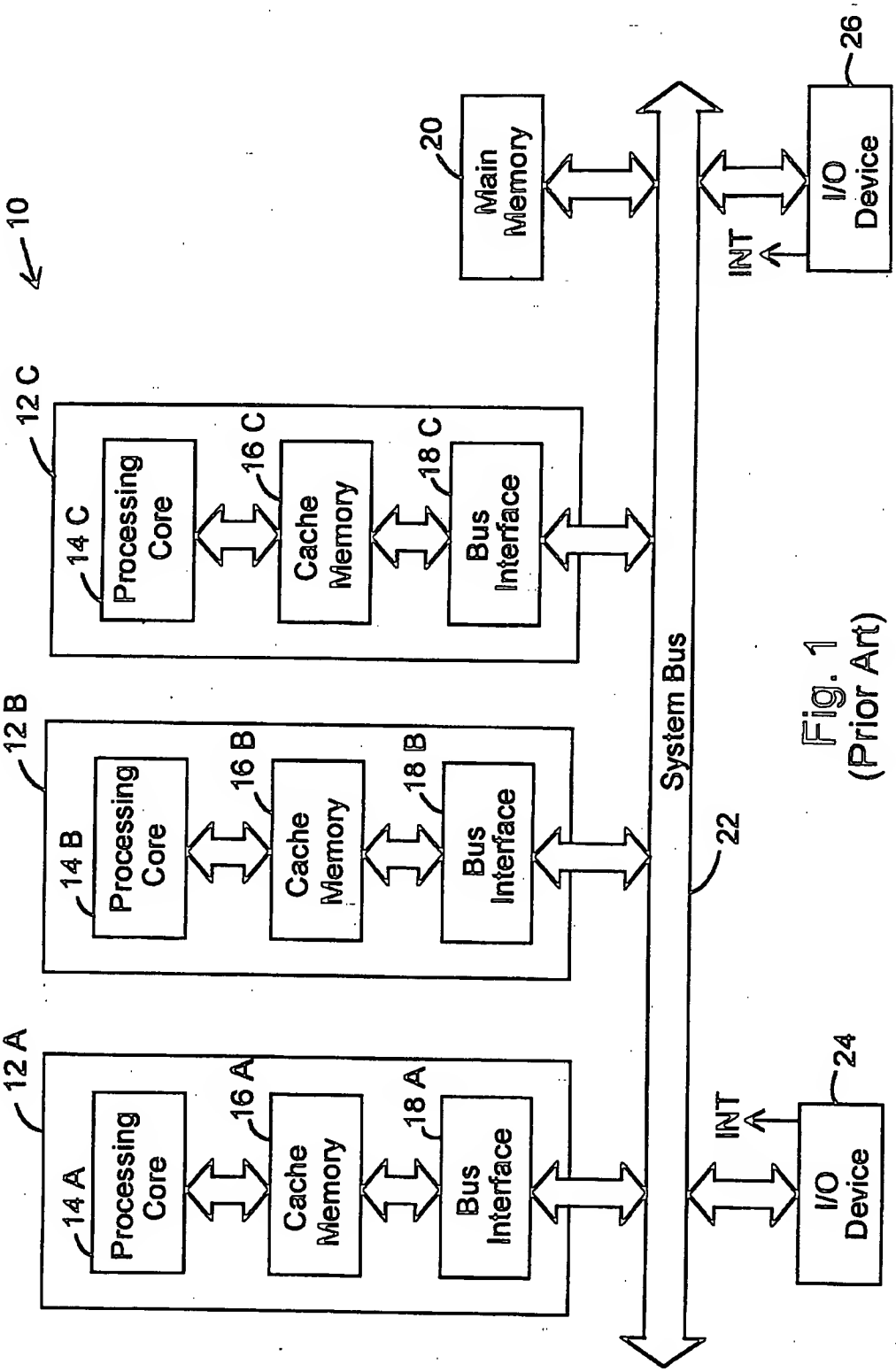


Fig. 1
(Prior Art)

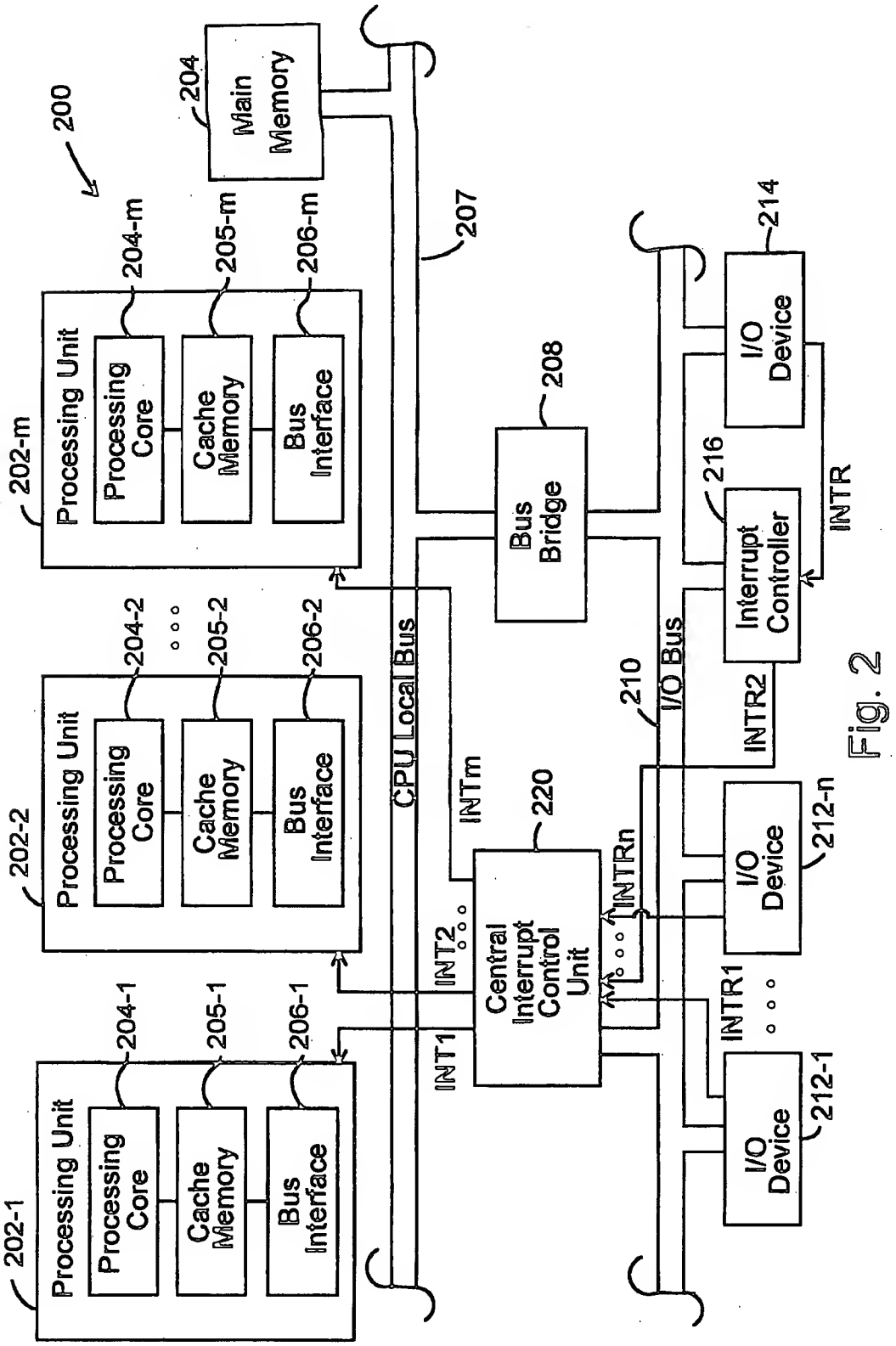

200
204
Main Memory
207
204-m
205-m
206-m
202-m
Processing Unit
Processing Core
Cache Memory
Bus Interface
208
Bus Bridge
214
I/O Device
216
Interrupt Controller
INTR
204-2
205-2
206-2
202-2
Processing Unit
Processing Core
Cache Memory
Bus Interface
CPU Local Bus
INTm
220
210
I/O Bus
INTR2
INTRn
INT2
Central Interrupt Control Unit
212-n
I/O Device
INTR1
204-1
205-1
206-1
202-1
Processing Unit
Processing Core
Cache Memory
Bus Interface
INT1
212-1
I/O Device

Fig. 2

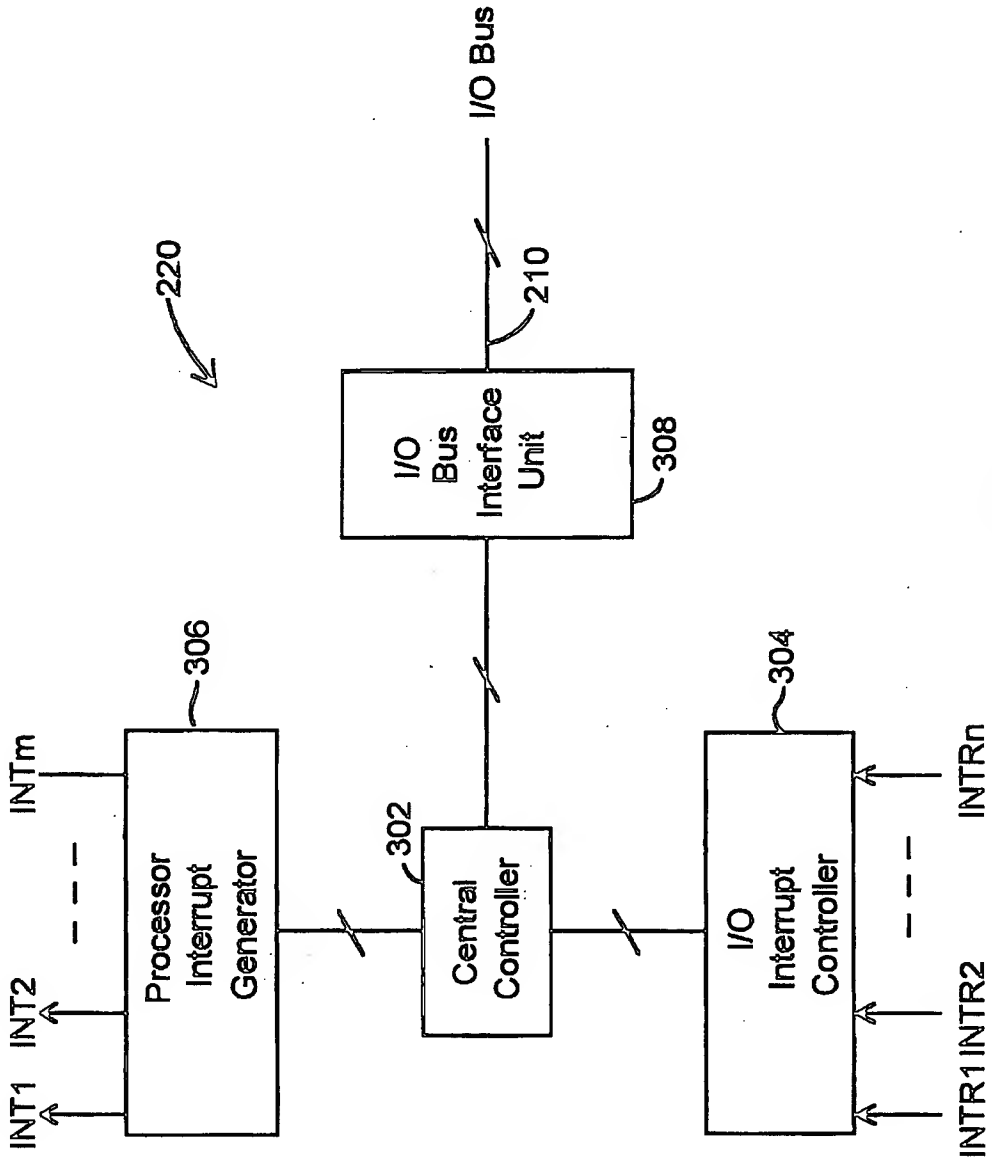

I/O Bus
220
210
I/O Bus Interface Unit
308
306
304
INTm
INT2
INT1
Processor Interrupt Generator
302
Central Controller
I/O Interrupt Controller
INTRn
INTR2
INTR1

Fig. 3

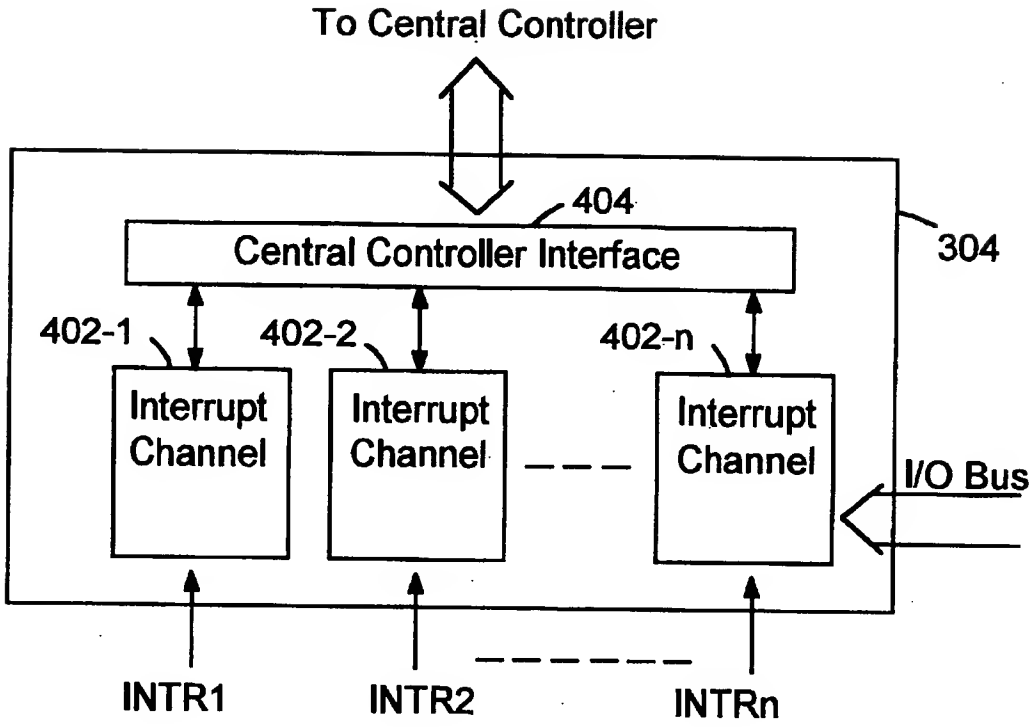

To Central Controller
404
Central Controller Interface
304
402-1
402-2
402-n
Interrupt Channel
Interrupt Channel
Interrupt Channel
I/O Bus
INTR1
INTR2
INTRn

Fig. 4

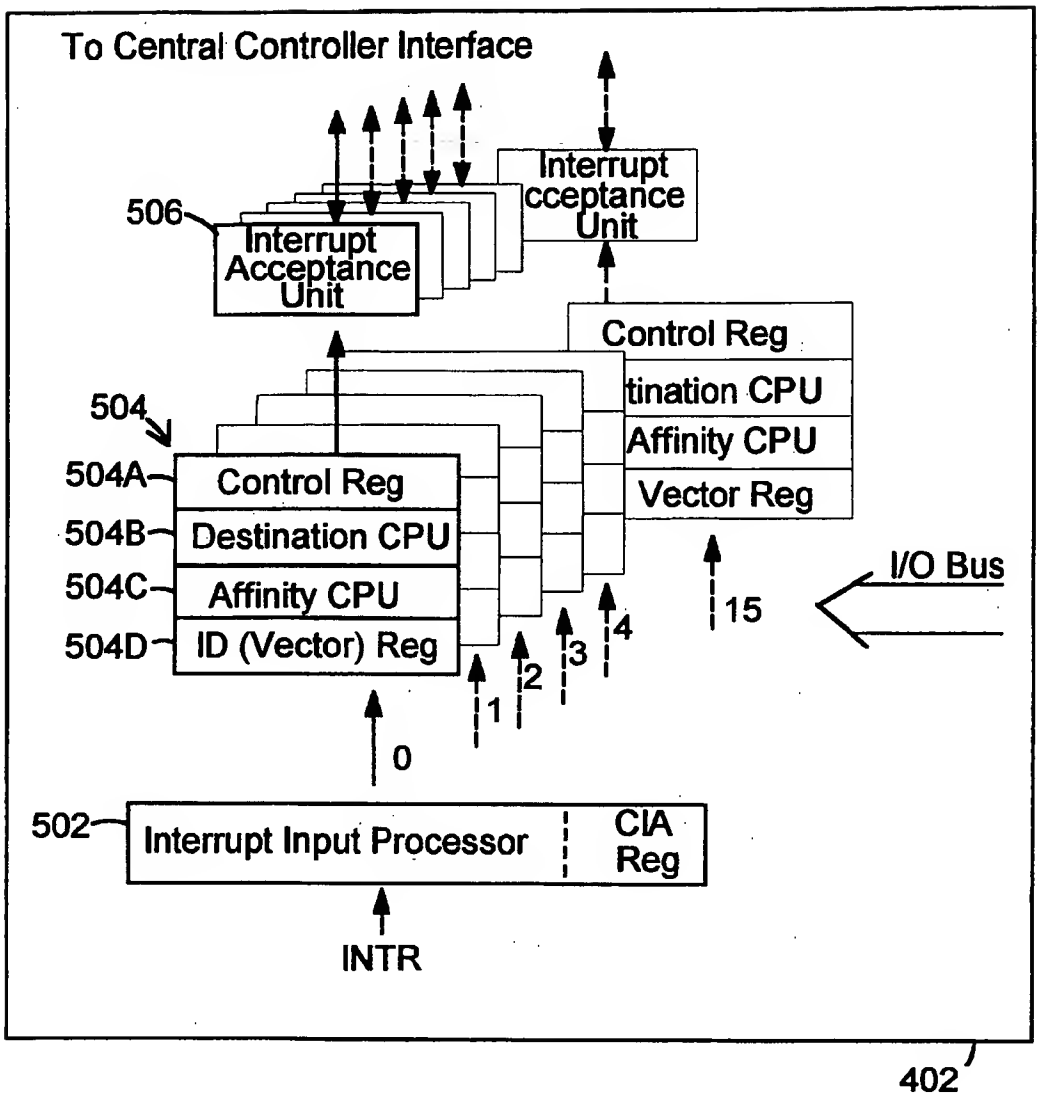

To Central Controller Interface
Interrupt cceptance Unit
506
Interrupt Acceptance Unit
Control Reg
tination CPU
Affinity CPU
Vector Reg
504
504A
Control Reg
504B
Destination CPU
504C
Affinity CPU
504D
ID (Vector) Reg
I/O Bus
15
4
3
2
1
0
502
Interrupt Input Processor
CIA Reg
INTR
402

Fig. 5

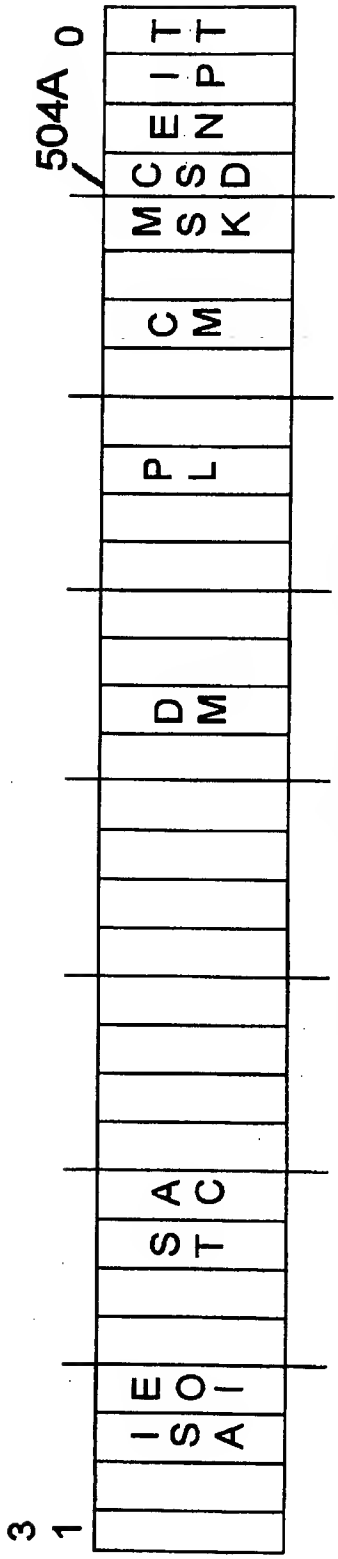

504A
0
3
1
MCSSDK
CM
PL
DM
SATC
IESOAI
Interrupt Channel Control Register

Fig. 6

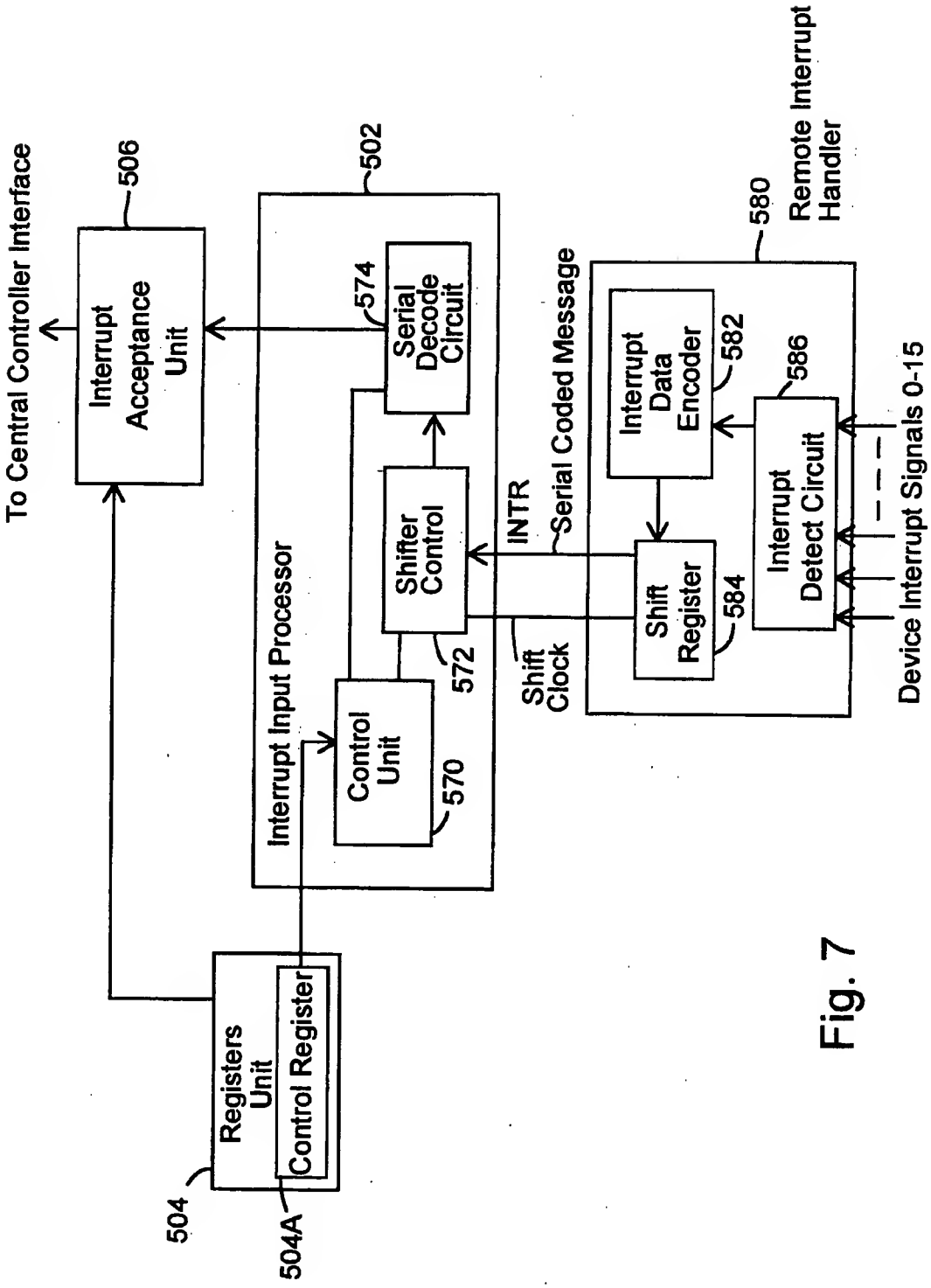

To Central Controller Interface
506
Interrupt Acceptance Unit
502
574
Serial Decode Circuit
Interrupt Input Processor
Shifter Control
572
Control Unit
570
INTR
Shift Clock
Serial Coded Message
580
Remote Interrupt Handler
Interrupt Data Encoder
582
586
Interrupt Detect Circuit
Shift Register
584
Device Interrupt Signals 0-15
Registers Unit
Control Register
504
504A

Fig. 7

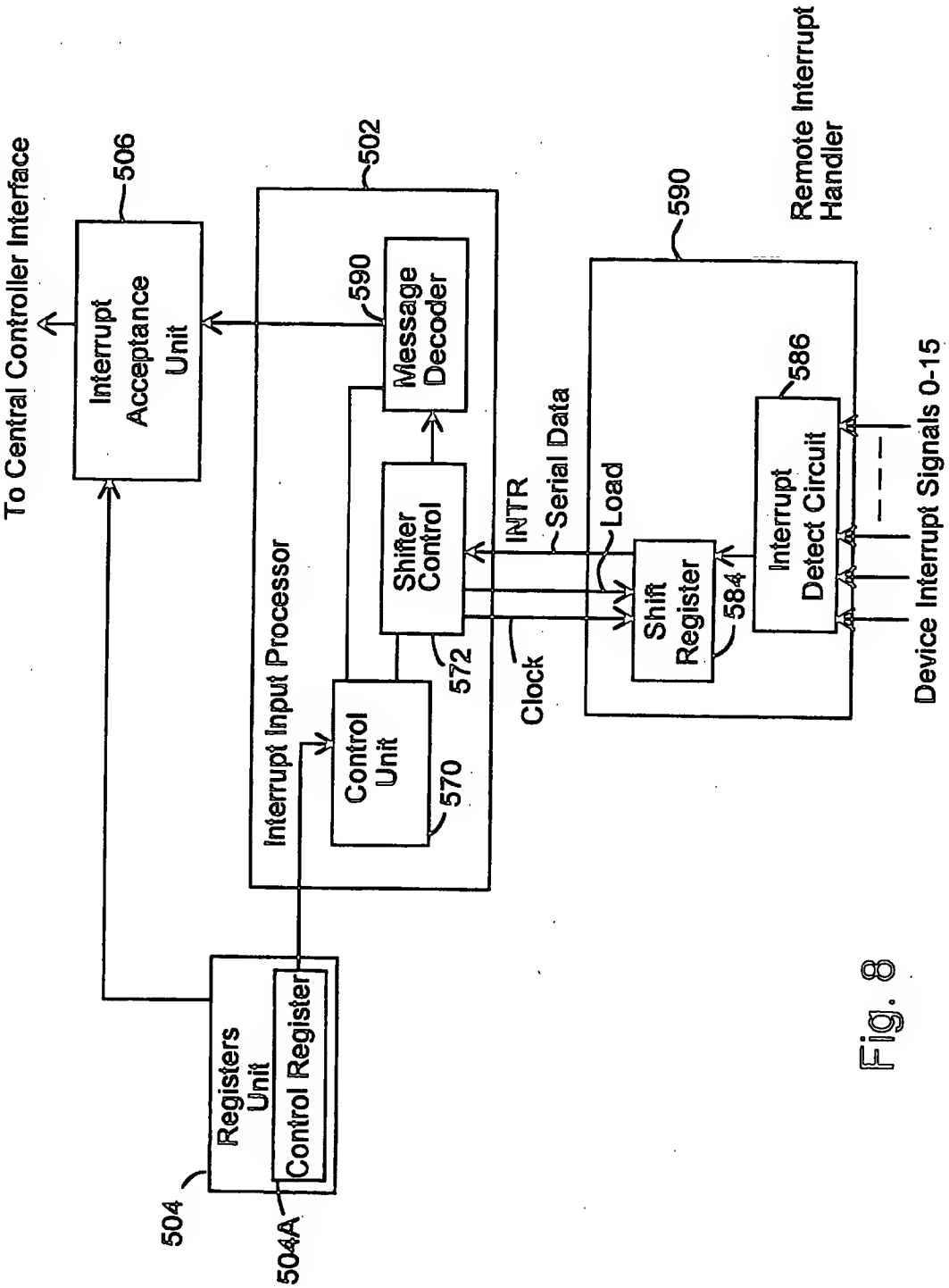

To Central Controller Interface
506
Interrupt Acceptance Unit
502
590
Message Decoder
Interrupt Input Processor
Shifter Control
572
Control Unit
570
INTR
Serial Data
Load
Clock
590
Remote Interrupt Handler
Shift Register
584
Interrupt Detect Circuit
586
Device Interrupt Signals 0-15
Registers Unit
Control Register
504
504A

Fig. 8

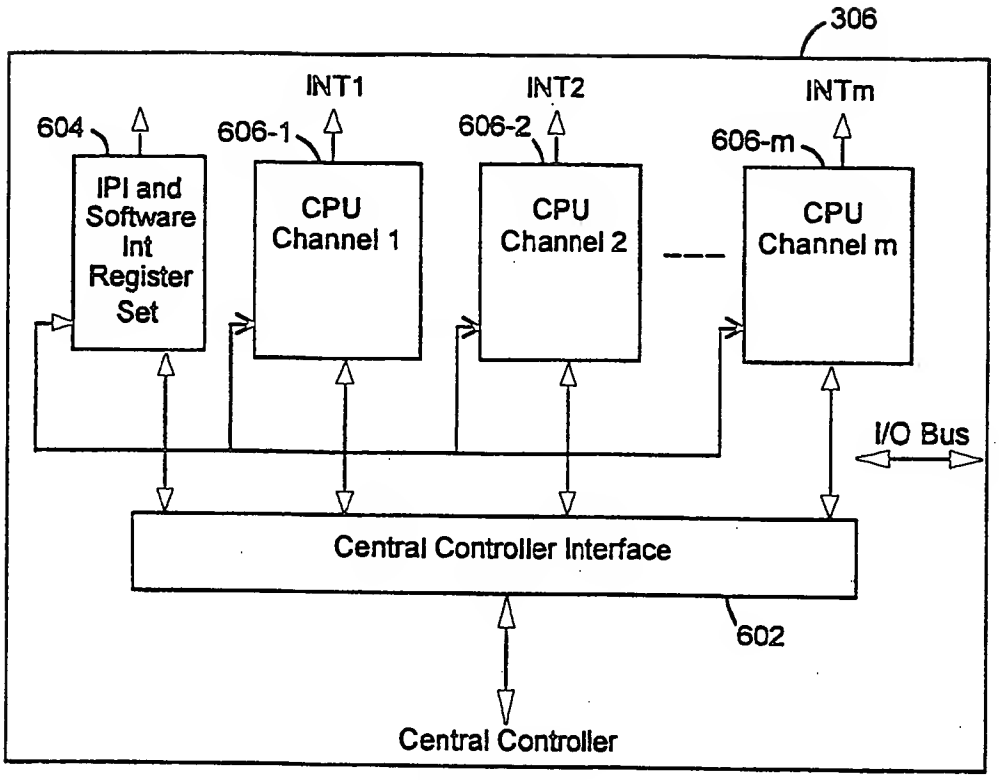

306
INT1
INT2
INTm
604
606-1
606-2
606-m
IPI and Software Int Register Set
CPU Channel 1
CPU Channel 2
CPU Channel m
I/O Bus
Central Controller Interface
602
Central Controller

Fig. 9

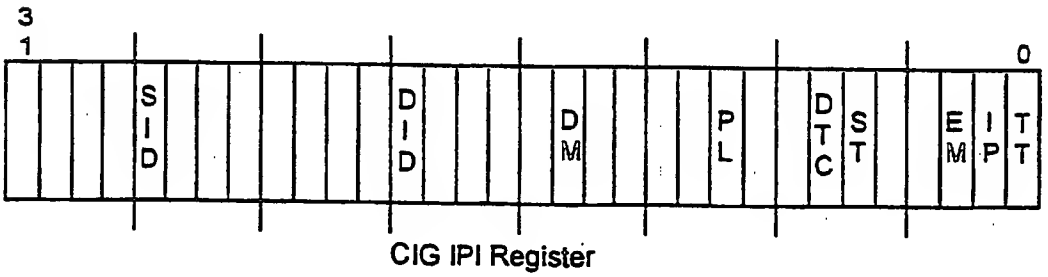

3
1
0
SID
DID
DM
PL
DTC
ST
EM
IP
TT
CIG IPI Register

Fig. 10B

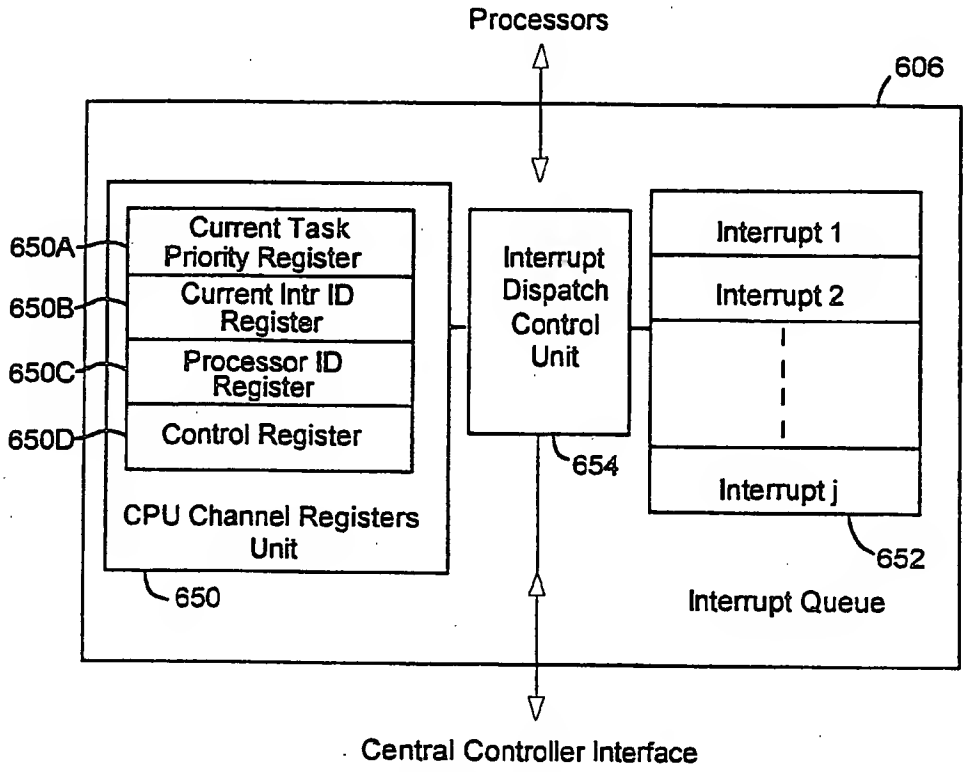


Fig. 10

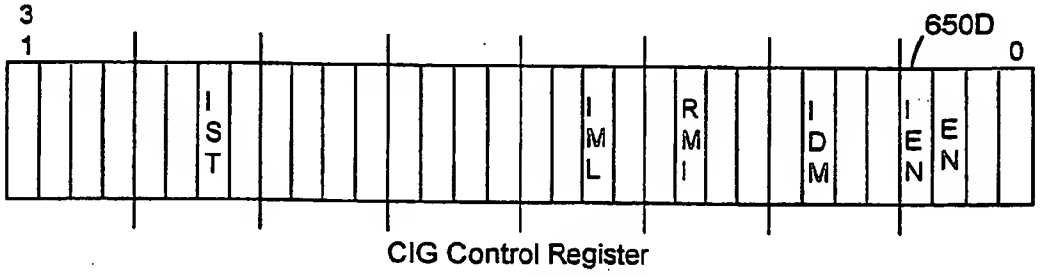


Fig. 10A

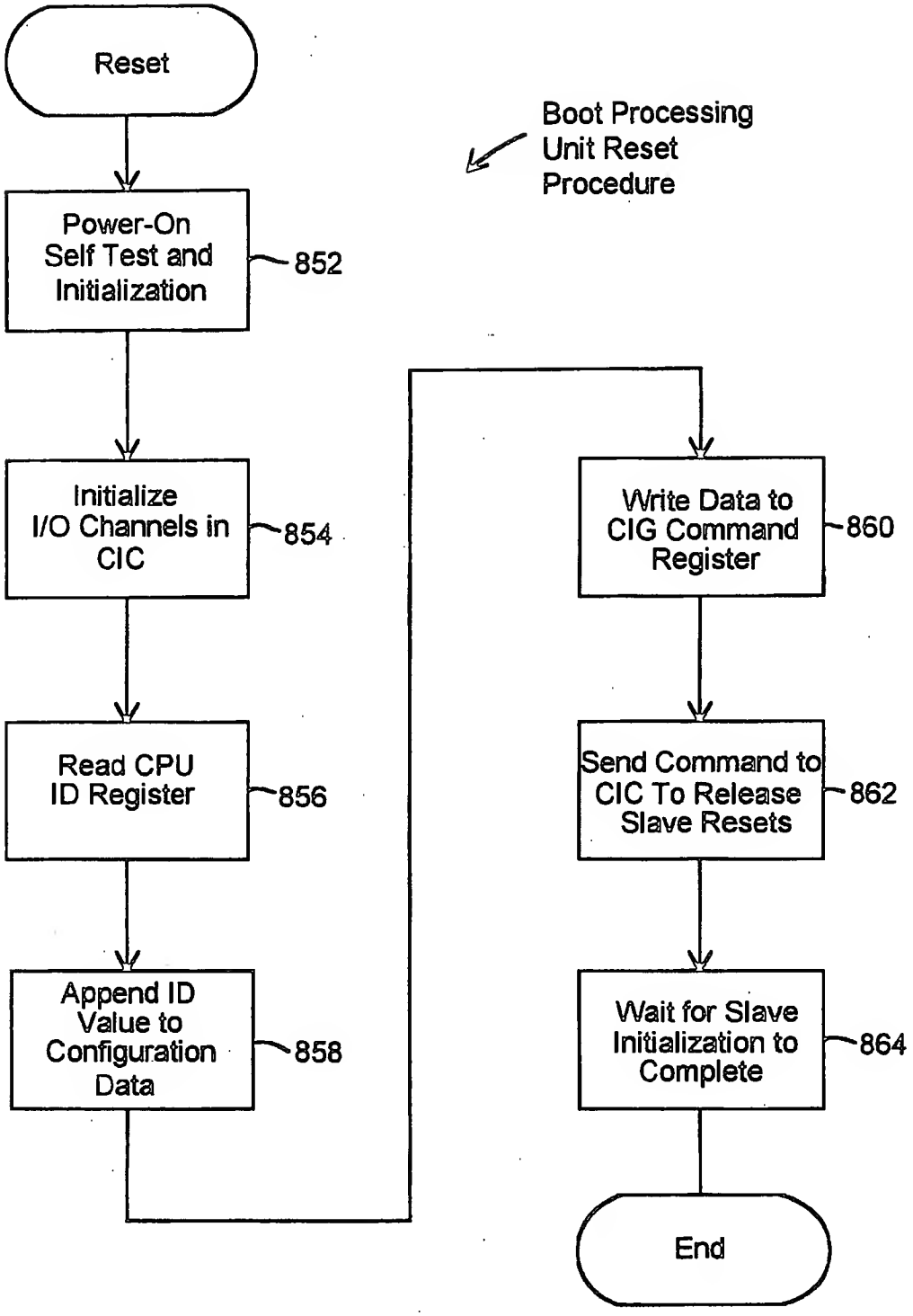

Reset
Boot Processing Unit Reset Procedure
Power-On Self Test and Initialization
852
Initialize I/O Channels in CIC
854
Read CPU ID Register
856
Append ID Value to Configuration Data
858
Write Data to CIG Command Register
860
Send Command to CIC To Release Slave Resets
862
Wait for Slave Initialization to Complete
864
End

Fig. 11

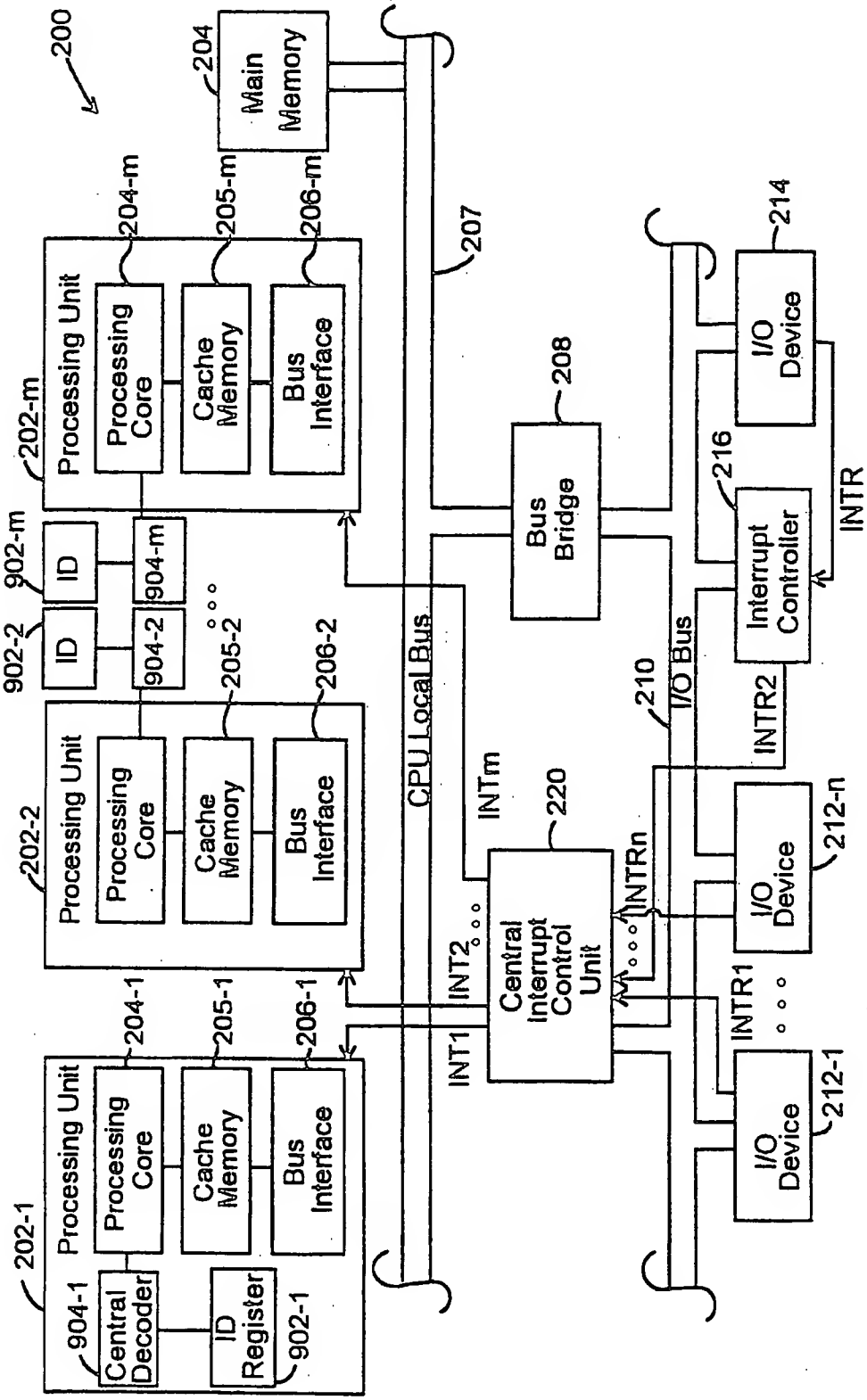

200
204
Main Memory
207
204-m
205-m
206-m
202-m
Processing Unit
Processing Core
Cache Memory
Bus Interface
902-m
ID
904-m
902-2
ID
904-2
205-2
206-2
202-2
Processing Unit
Processing Core
Cache Memory
Bus Interface
CPU Local Bus
INTm
INT2
INT1
220
Central Interrupt Control Unit
INTRn
INTR2
INTR1
208
Bus Bridge
210
I/O Bus
216
Interrupt Controller
INTR
214
I/O Device
212-n
I/O Device
212-1
I/O Device
204-1
205-1
206-1
202-1
Processing Unit
Processing Core
Cache Memory
Bus Interface
904-1
Central Decoder
ID Register
902-1

Fig. 12

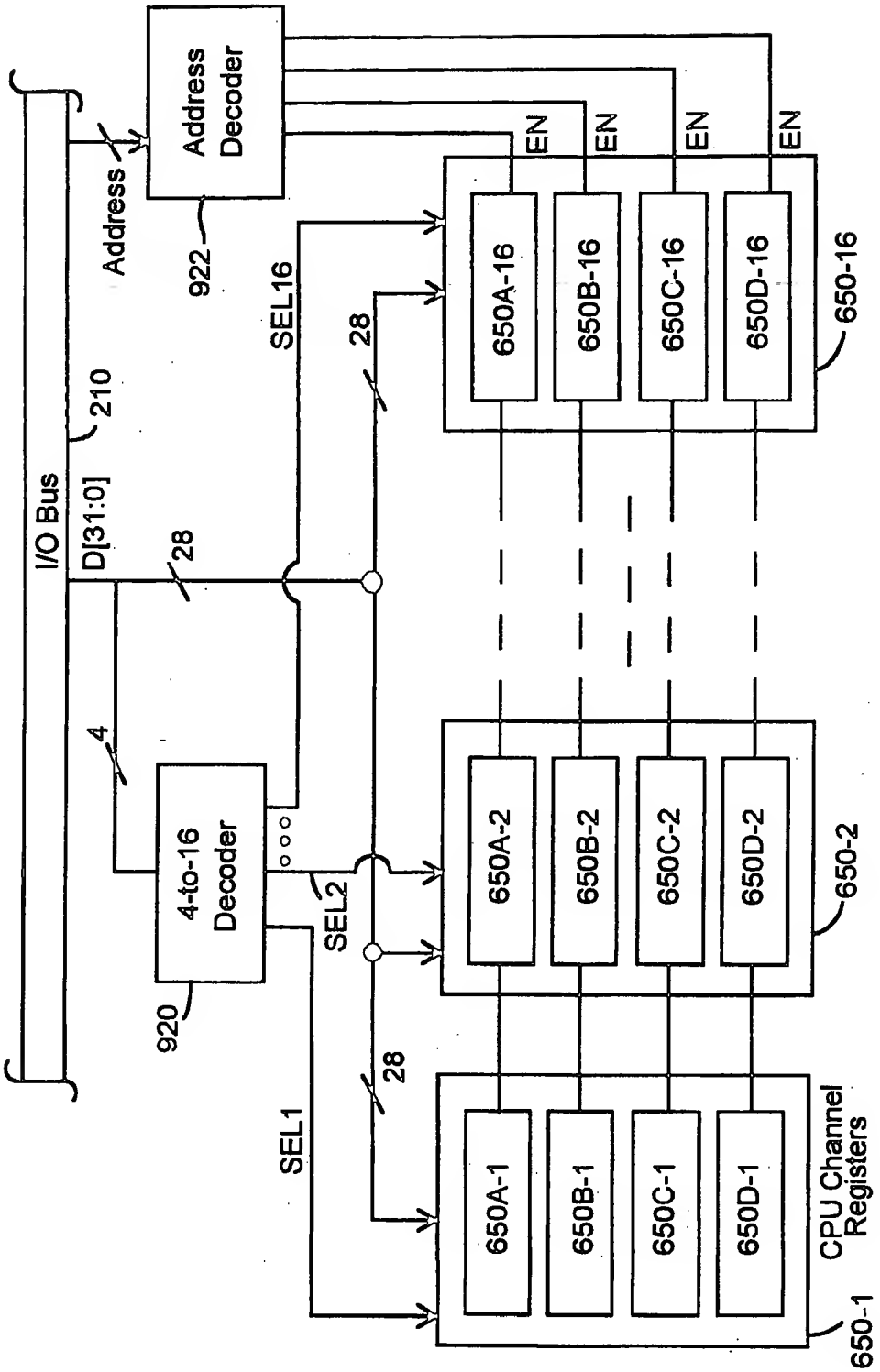

Address Decoder
922
Address
I/O Bus
210
D[31:0]
28
SEL16
28
650A-16
650B-16
650C-16
650D-16
650-16
EN
EN
EN
EN
4
4-to-16 Decoder
920
SEL2
650A-2
650B-2
650C-2
650D-2
650-2
SEL1
28
650A-1
650B-1
650C-1
650D-1
CPU Channel Registers
650-1

Fig. 13

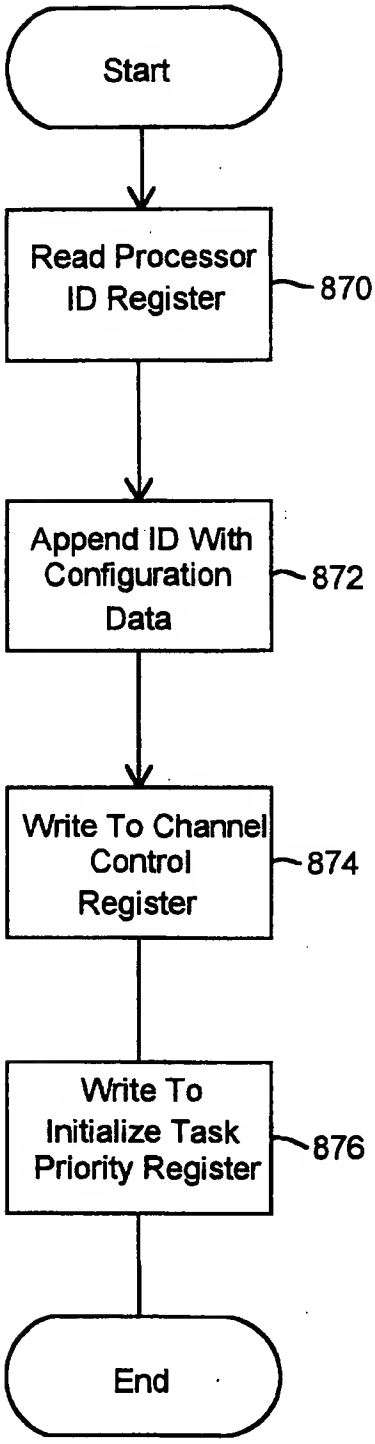

Start
Read Processor ID Register
870
Append ID With Configuration Data
872
Write To Channel Control Register
874
Write To Initialize Task Priority Register
876
End

Fig. 14

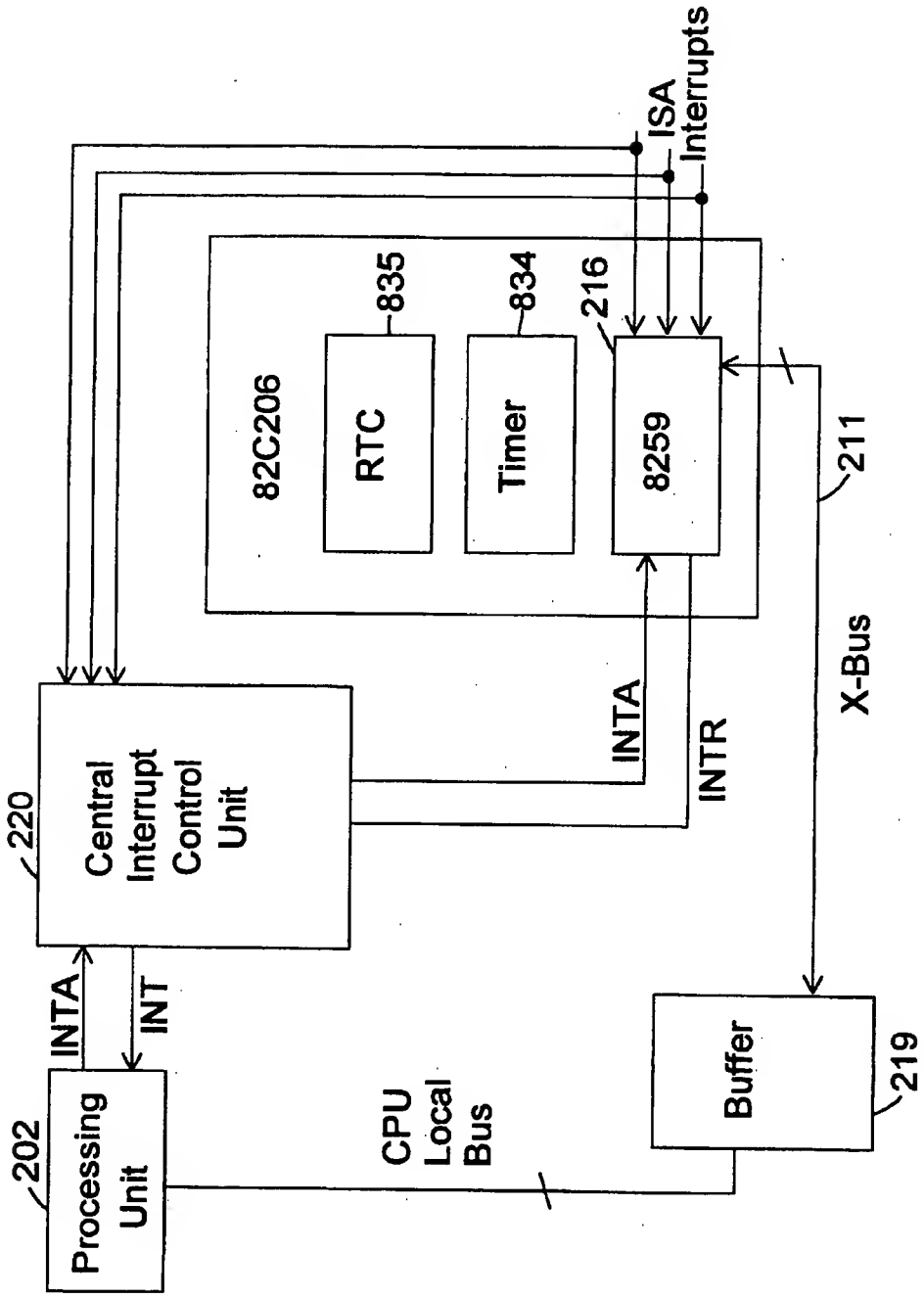

ISA Interrupts
82C206
RTC
835
Timer
834
8259
216
211
X-Bus
INTA
INTR
Central Interrupt Control Unit
220
INTA
INT
Processing Unit
202
CPU Local Bus
Buffer
219

Fig. 15

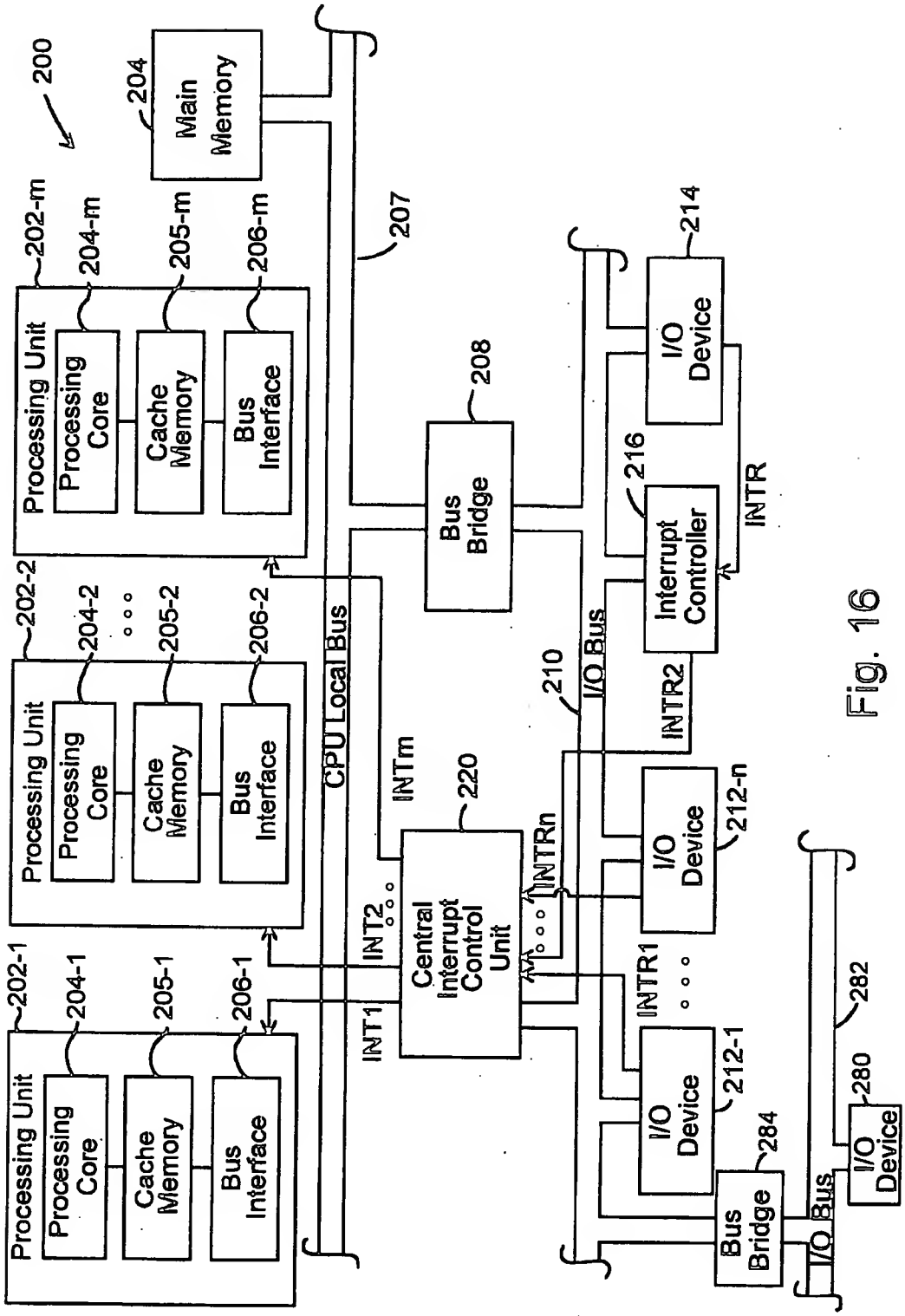

200
204
Main Memory
207
202-m
204-m
205-m
206-m
Processing Unit
Processing Core
Cache Memory
Bus Interface
208
Bus Bridge
214
I/O Device
216
Interrupt Controller
INTR
202-2
204-2
205-2
206-2
CPU Local Bus
I/O Bus
210
INTm
INTR2
220
INT2
INTRn
Central Interrupt Control Unit
212-n
INT1
INTR1
202-1
204-1
205-1
206-1
212-1
282
284
280

Fig. 16

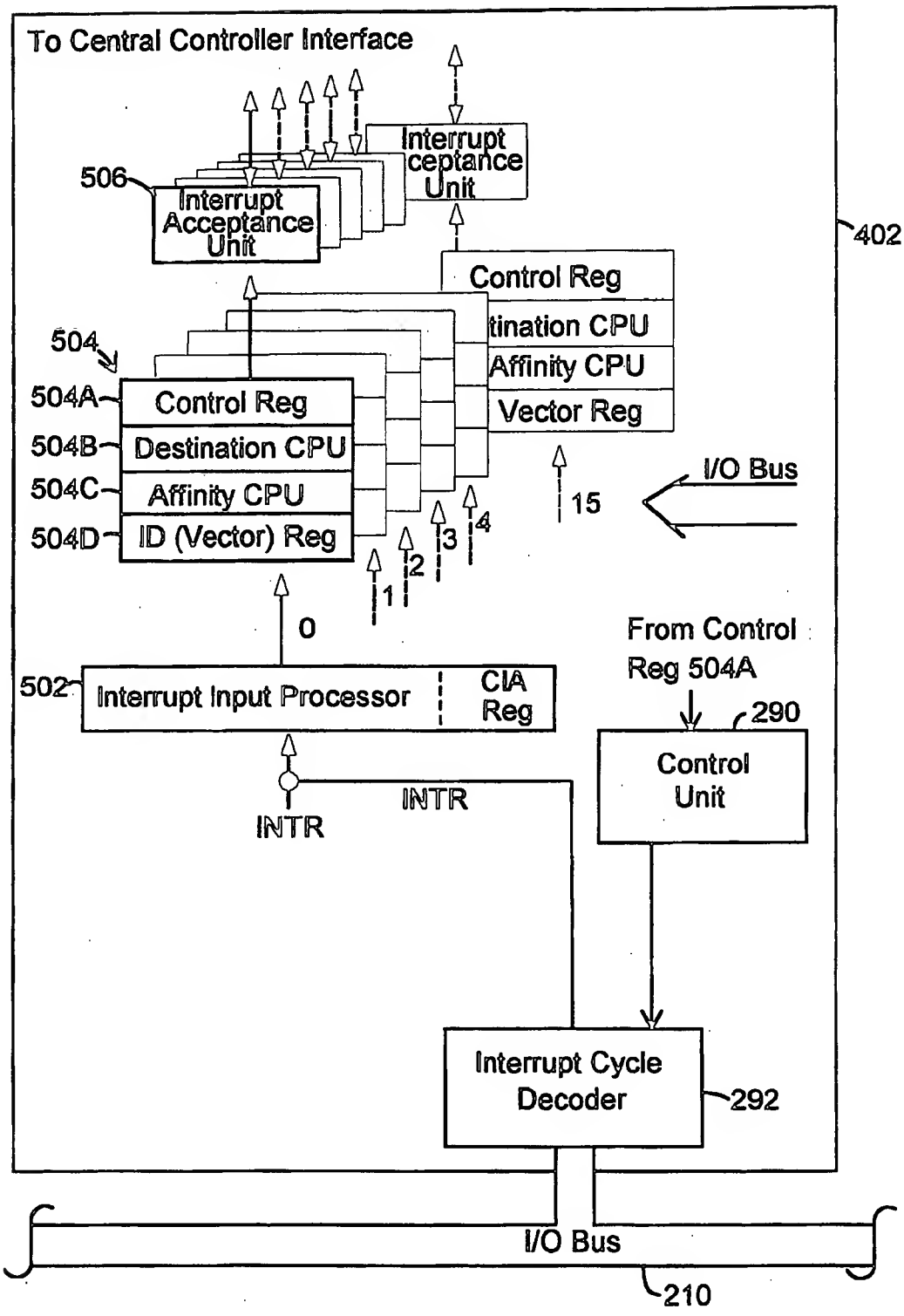

To Central Controller Interface
Interrupt ceptance Unit
506
Interrupt Acceptance Unit
402
Control Reg
tination CPU
Affinity CPU
Vector Reg
504
504A
Control Reg
504B
Destination CPU
504C
Affinity CPU
504D
ID (Vector) Reg
I/O Bus
15
4
3
2
1
0
From Control Reg 504A
290
502
Interrupt Input Processor
CIA Reg
Control Unit
INTR
INTR
Interrupt Cycle Decoder
292
I/O Bus
210

Fig. 17

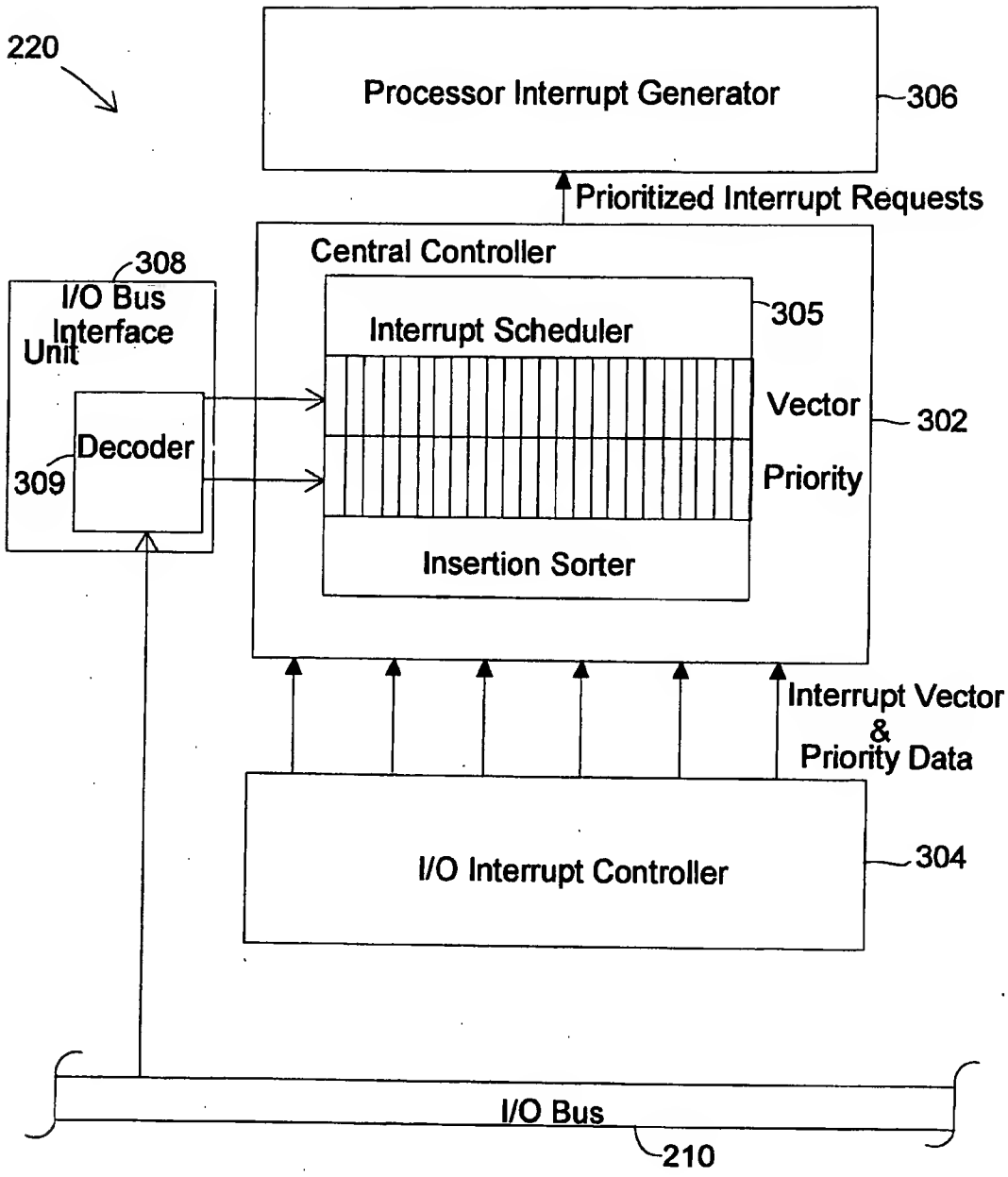

220
Processor Interrupt Generator
306
Prioritized Interrupt Requests
Central Controller
308
I/O Bus Interface Unit
Interrupt Scheduler
305
Vector
302
Priority
Decoder
309
Insertion Sorter
Interrupt Vector & Priority Data
I/O Interrupt Controller
304
I/O Bus
210

Fig. 18

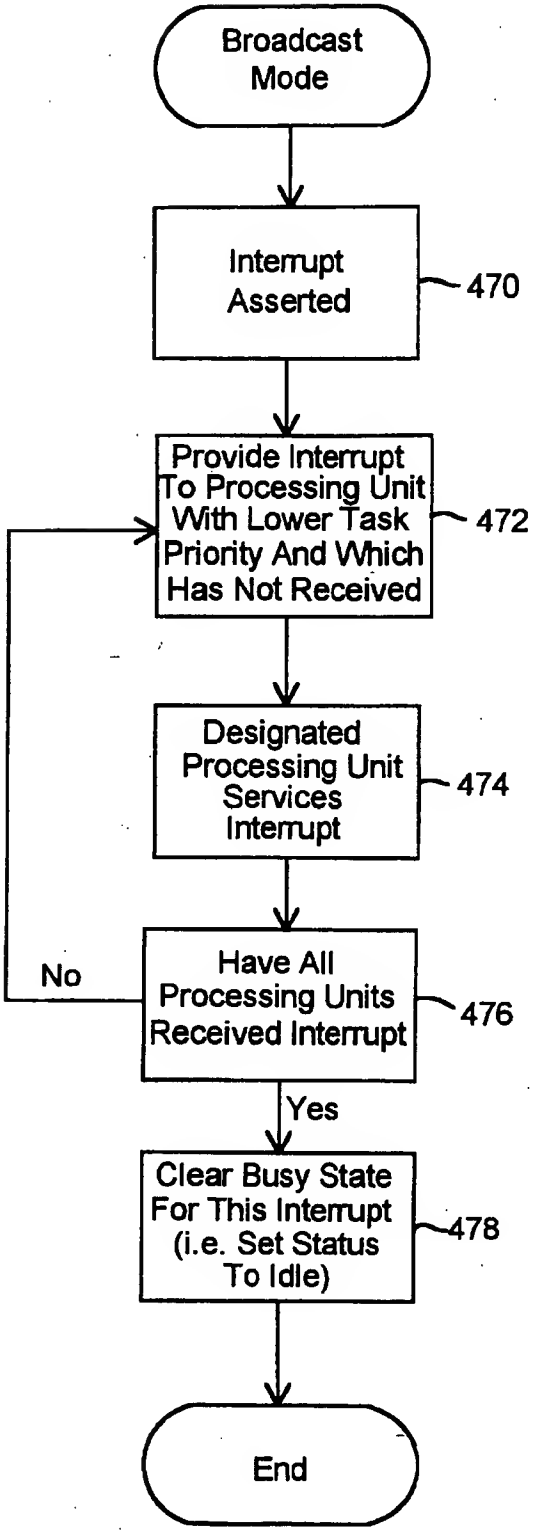

Broadcast Mode
Interrupt Asserted
470
Provide Interrupt To Processing Unit With Lower Task Priority And Which Has Not Received
472
Designated Processing Unit Services Interrupt
474
Have All Processing Units Received Interrupt
476
No
Yes
Clear Busy State For This Interrupt (i.e. Set Status To Idle)
478
End

Fig. 19

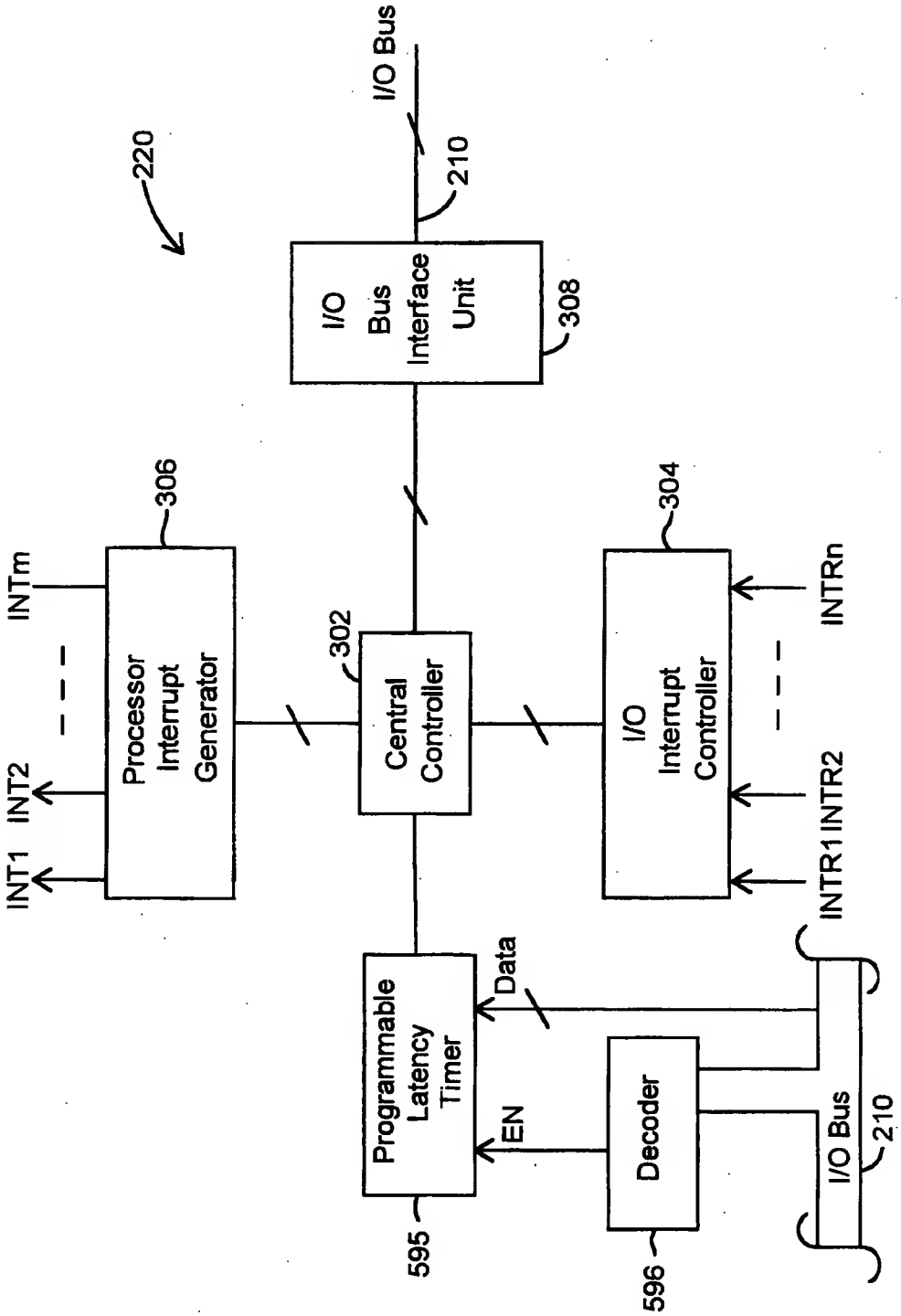

220
I/O Bus
210
I/O Bus Interface Unit
308
306
304
INTm
INT2
INT1
Processor Interrupt Generator
302
Central Controller
I/O Interrupt Controller
INTRn
INTR2
INTR1
Data
Programmable Latency Timer
EN
Decoder
I/O Bus
210
595
596

Fig. 20

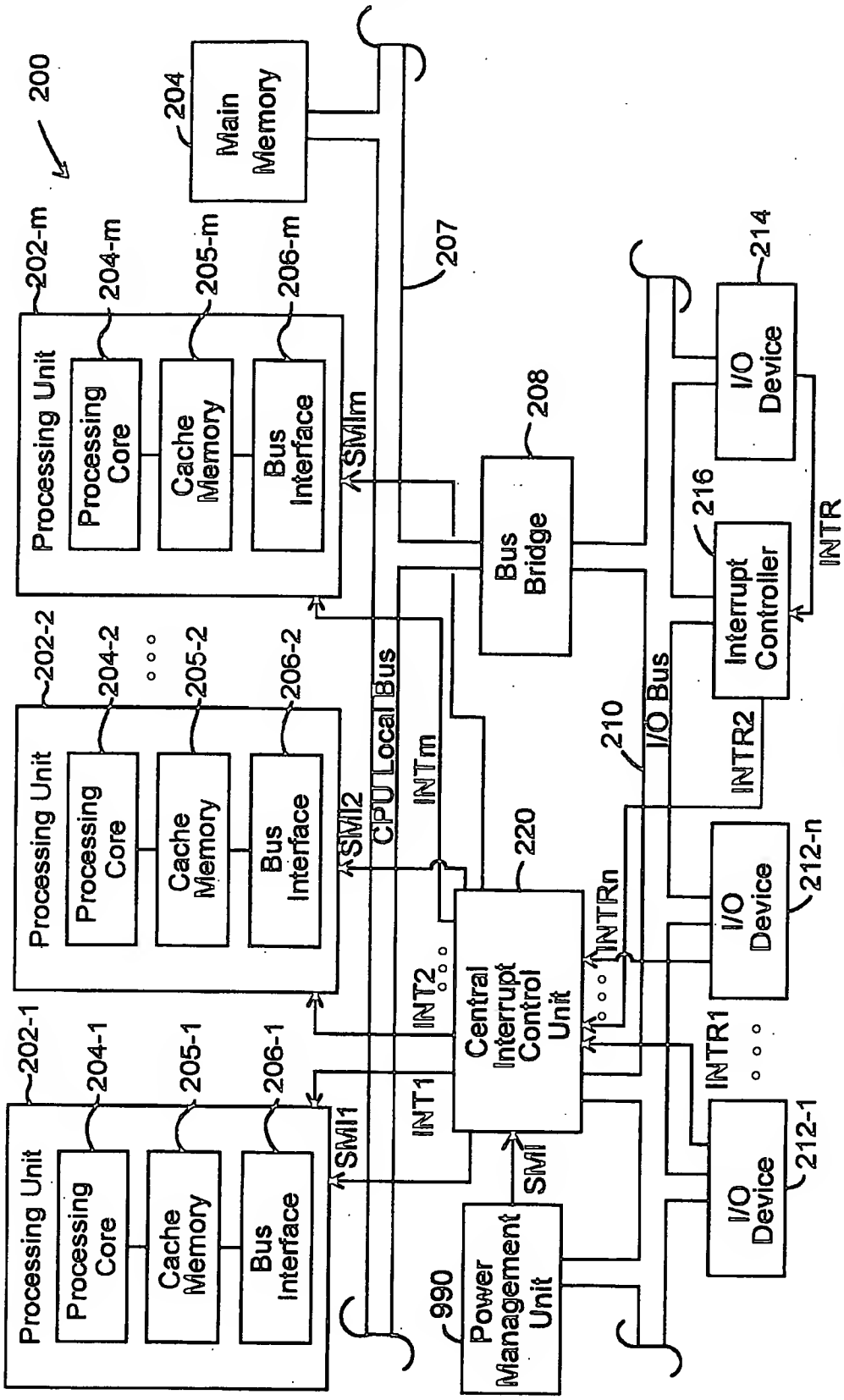

200
204
Main Memory
202-m
204-m
205-m
206-m
Processing Unit
Processing Core
Cache Memory
Bus Interface
SMIm
207
208
Bus Bridge
214
I/O Device
216
Interrupt Controller
INTR
202-2
204-2
205-2
206-2
Processing Unit
Processing Core
Cache Memory
Bus Interface
SMI2
CPU Local Bus
INTm
210
I/O Bus
INTR2
220
202-1
204-1
205-1
206-1
Processing Unit
Processing Core
Cache Memory
Bus Interface
SMI1
INT2
INT1
Central Interrupt Control Unit
INTRn
212-n
I/O Device
INTR1
212-1
I/O Device
SMI
990
Power Management Unit

Fig. 21

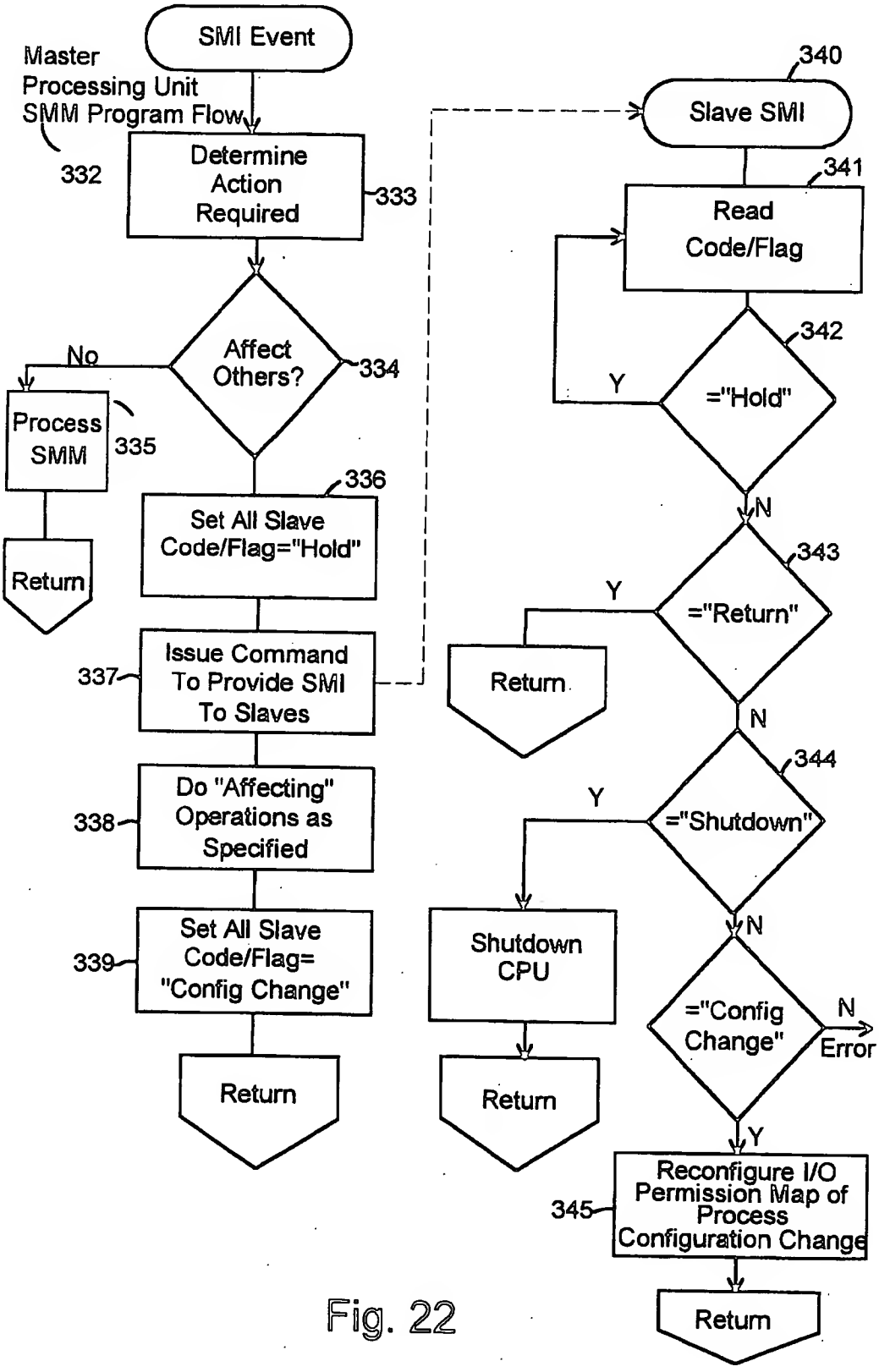

SMI Event
Master Processing Unit SMM Program Flow
332
Determine Action Required
333
Affect Others?
334
No
Process SMM
335
Return
336
Set All Slave Code/Flag="Hold"
337
Issue Command To Provide SMI To Slaves
338
Do "Affecting" Operations as Specified
339
Set All Slave Code/Flag= "Config Change"
Return
340
Slave SMI
341
Read Code/Flag
342
="Hold"
Y
N
343
="Return"
Y
Return
N
344
="Shutdown"
Y
Shutdown CPU
Return
N
="Config Change"
N
Error
Y
345
Reconfigure I/O Permission Map of Process Configuration Change
Return

Fig. 22

INTERRUPT HANDLING MECHANISM TO PREVENT SPURIOUS INTERRUPTS IN A SYMMETRICAL MULTIPROCESSING SYSTEM

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to computer systems and more particularly to interrupt control architectures and schemes employed within symmetrical multiprocessing systems.

2. Description of the Relevant Art

Computer systems employing multiple processing units hold a promise of economically accommodating performance capabilities that surpass those of current single-processor based systems. Within a multiprocessing environment, rather than concentrating all the processing for an application in a single processor, tasks are divided into groups or "threads" that can be handled by separate processors. The overall processing load is thereby distributed among several processors, and the distributed tasks may be executed simultaneously in parallel. The operating system software divides various portions of the program code into the separately executable threads, and typically assigns a priority level to each thread.

FIG. **1** is a block diagram of a so-called symmetrical multiprocessing system **10** including a plurality of processing units **12A–12C**. Each processing unit **12A–12C** includes a processing core **14A–14C**, a cache memory **16A–16C**, and a bus interface **18A–18C**, respectively. The processing units **12A–12C** are coupled to a main memory **20** via a system bus **22**. A pair of I/O devices **24** and **26** are further coupled to system bus **22**.

The multiprocessing system **10** of FIG. **1** is symmetrical in the sense that all processing units **12A–12C** share the same memory space (i.e., main memory **20**) and access the memory space using the same address mapping. The multiprocessing system **10** is further symmetrical in the sense that all processing units **12A–12C** share equal access to the same I/O subsystem.

In general, a single copy of the operating system software as well as a single copy of each user application file is stored within main memory **20**. Each processing unit **12A–12C** executes from these single copies of the operating system and user application files. Although processing cores **14A–14C** may be executing code simultaneously, it is noted that only one of the processing units **12A–12C** may assume mastership of the system bus **22** at a given time. Thus, a bus arbitration mechanism (not shown) is provided to arbitrate concurrent bus requests of two or more processing units and to grant mastership to one of the processing units based on a predetermined arbitration algorithm. A variety of bus arbitration techniques are well-known.

The high speed cache memory **16A–16C** of each processing unit **12A–12C**, respectively, stores data most recently accessed by the respective processing unit along with address tags that indicate the main memory address to which the associated data corresponds. Since programs tend to execute the same sections of code and access the same data structures repeatedly, many of the locations accessed will already be stored in the cache if the cache is sufficiently large.

The cache mechanisms provide two significant benefits. First, because the caches are implemented with high-speed memory and can be accessed without bus arbitration and buffer delays, an access to a location stored in a respective cache is much faster than a main memory access. Second, because an access to a location stored in the respective cache does not require access to the system bus, the bus utilization of each processor is greatly reduced. The system bus is therefore available to service other requested transactions. Typically, the higher the "hit rate", the better the overall system performance. The hit rate is the percentage of accesses by a particular processing core that are to locations already stored in the cache. Well designed systems with moderately large caches can achieve hit rates of over ninety percent.

An important consideration with respect to multiprocessing systems that employ cache memories is data coherency. Since multiple copies of the data (and instructions) stored by main memory **20** may concurrently reside in one or more of the cache memories **16A–16C**, a specialized mechanism must be employed to maintain the integrity of data in the event that one of the memory subsystems is updated (i.e., written with new data). For example, consider a situation wherein a particular section of data is updated within cache memory **16A** by processing core **14A** but is not updated within the corresponding section of main memory **20**. If processing core **14B** subsequently accesses the same section of code, there must be some reliable mechanism to track which section is up-to-date and which section is no longer valid to ensure that processing core **14B** accesses the proper data. A variety of techniques have therefore been developed with the goal of efficiently maintaining cache coherency, including those based on so-called write-through and write-back techniques. Various cache coherency techniques are described within a host of publications of the known prior art, and are not discussed further herein.

Another important consideration with respect to symmetrical multiprocessing systems is the handling and distribution of interrupts generated by various system resources. For example, in the system of FIG. 1, I/O devices **24** and **26** may each assert a respective interrupt signal based on the occurrence (or non-occurrence) of a particular event. As will be appreciated by those of skill in the art, interrupts are routinely generated by system resources such as keyboard devices, printers, and timers, among other things. Many systems also accommodate software interrupts whereby an interrupt may be asserted in response to software command. Due to the number of different interrupts that may occur within a system, it is desirable to provide a mechanism to efficiently manage and distribute the interrupts to achieve optimal system performance and bus utilization.

One technique for handling interrupts employs a centralized interrupt controller that is capable of receiving a plurality of interrupts and of prioritizing and distributing the interrupts amongst the various processing units. A problem associated with a centralized interrupt control technique is that the total number of interrupts which can be accommodated is typically limited by the number of input pins provided to the centralized interrupt controller. In other words, for example, if the centralized interrupt controller includes a total of sixteen interrupt input pins, greater than sixteen interrupt generating devices typically cannot be accommodated within the system. This limits system flexibility. Furthermore, within such systems a dedicated interrupt line must be connected to each interrupt source. Such dedicated interrupt lines may not be available on remote cabling networks that connect one or more peripherals to the computer system. Again, system flexibility may consequently be limited.

Another problem which may be encountered with respect to interrupt management within symmetrical multiprocess-

ing systems is the occurrence of invalid or "spurious" interrupts. As is generally known, when a level triggered interrupt is being serviced by a designated processing unit, an I/O command is typically associated with the interrupt service routine that, where executed, causes the interrupt source to deassert the interrupt signal. Subsequently, an End Of Interrupt (EOI) command is executed to inform the interrupt controller that the interrupt service has completed. A spurious interrupt may occur if a significant latency is introduced between the time at which the processing unit executes the I/O command (to cause the interrupt source to deassert the interrupt signal) and the time at which the interrupt signal is actually deasserted. Such a latency may occur, for example, if the I/O device resides on a remote bus via several bus bridge units. If the interrupt source does not deassert the interrupt signal before the centralized interrupt controller responds to the End Of Interrupt command, the continued assertion of the interrupt may be detected by the interrupt controller, thus causing the interrupt to unintentionally re-initiate.

The management of timer tick interrupts poses another problem within symmetrical multiprocessing systems. As is generally known, in a multiprocessing system there are some interrupts which need to be handled by all processing units in a broadcast fashion. For example, many multiprocessing systems use the timer tick interrupt to signal the end of a time slice and to thus cause each processing unit to execute a task switch. This is typically accomplished by either simultaneously sending the timer tick interrupt to all processing units or by sending the timer tick interrupt to the first processing unit in the chain and passing it to the others via interprocessor interrupts (IPIs). However, there are drawbacks to both of these schemes. If all processing units are interrupted at the same time, they will attempt to access the same shared interrupt handler code and will lock shared system resources at the same time. This may cause a great deal of contention and may require some processing units to wait until the required resources become available. If the timer tick interrupt is provided to the first processing unit in the chain and is passed to the others by using interprocessor interrupts, increased software overhead and bus utilization is incurred.

Still additional problems related to interrupt management within symmetrical multiprocessing systems include the integration of system management interrupts (SMI) and the prioritization of interrupts. It is desirable to provide mechanisms within symmetrical multiprocessing systems that allow both efficient integration of SMI interrupts within the system as well as flexible interrupt prioritization.

SUMMARY OF THE INVENTION

The problems outlined above are in large part solved by a symmetrical multiprocessing system in accordance with the present invention. In one embodiment, a symmetrical multiprocessing system is provided that includes centralized interrupt control unit. The interrupt control unit is coupled to a plurality of processing units and to a plurality of interrupt sources. The interrupt control unit advantageously allows for the expansion of each interrupt pin by setting the interrupt control unit in a cascade mode. Furthermore, the central control unit is responsive to specialized interrupt cycles which allows I/O devices and/or bus bridge devices to initiate an interrupt without requiring a dedicated interrupt line. The central interrupt control unit further allows each interrupt to be prioritized independently of its associated vector ID, and prevents the occurrence of spurious interrupts by providing a programmable latency timer which causes the central interrupt control unit to delay its response to End Of Interrupt (EOI) instructions. An auto-chaining technique is further implemented by the central interrupt control unit to sequentially provide broadcast interrupts to various processing units based on their current task priority values. Finally, the central interrupt control unit further handles system management interrupts (SMIs) from sources such as power management units and ensures proper system operation even if the requested system management function affects operations being carried on by other processing units.

Broadly speaking, the present invention contemplates an interrupt controller comprising a plurality of interrupt input terminals, a central controller coupled to the input terminals for prioritizing the interrupt signals and for providing the interrupt signals to at least one processing unit, and a status register coupled to the central controller for storing information indicative of whether a particular interrupt signal is idle, wherein the central controller controls the status of the status register according to interrupt signals dispatched to the processing unit and an End Of Interrupt command executed by the processing unit. The interrupt controller further comprises a programmable latency timer coupled to the central controller, wherein a value within the programmable latency timer controls a time at which the central controller updates the status register in response to an End Of Interrupt command.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

FIG. **1** is a block diagram of a typical symmetrical multiprocessing system including a plurality of processing units.

FIG. **2** is a bock diagram of a symmetrical multiprocessing system including a centralized interrupt controller mechanism according to one embodiment of the present invention.

FIG. **3** is a block diagram of a central interrupt control unit.

FIG. **4** is a block diagram of an I/O interrupt controller.

FIG. **5** is a block diagram illustrative of an interrupt channel.

FIG. **6** is a diagram of an interrupt channel control register and its associated fields.

FIG. **7** is a block diagram that illustrates the hardware associated with the symmetrical multiprocessing system during one cascading mode of an interrupt input processor.

FIG. **8** is a block diagram that illustrates hardware associated with another cascading mode of the symmetrical multiprocessing system.

FIG. **9** is a block diagram of a processor interrupt generator.

FIG. **10** is a block diagram illustrative of a processor channel.

FIG. **10A** is a diagram that illustrates a CPU channel control register along with its associated fields.

FIG. **10B** is a diagram that illustrates an interprocessor interrupt register along with its associated fields.

FIG. **11** is a flow diagram which illustrates the operation of the boot processing unit upon system reset.

FIG. **12** is a diagram that illustrates hardware which enables the CPU channel registers unit of each CPU channel.

FIG. **13** is a block diagram that illustrates circuitry embodied within the central interrupt control unit.

FIG. **14** is a flow diagram that depicts the initialization sequence for each slave processing unit.

FIG. **15** is a block diagram which illustrates one of the processing units coupled to the central interrupt control unit.

FIG. **16** is a block diagram of a multiprocessing system which accommodates specially defined interrupt cycles for initiating an interrupt.

FIG. **17** is a block diagram that illustrates an interrupt channel including hardware configured to decode an interrupt cycle.

FIG. **18** is a block diagram that illustrates a portion of the central interrupt control unit including an internal portion of the central controller.

FIG. **19** is a flow diagram that illustrates the delivery of an interrupt during the broadcast mode.

FIG. **20** is a block diagram of the central control unit including a programmable latency timer.

FIG. **21** is a block diagram of a symmetrical multiprocessing system including a power management unit capable of asserting a system management interrupt.

FIG. **22** is a flow diagram that depicts operation of the symmetrical multiprocessing system when a system management interrupt is asserted.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE INVENTION

Referring next to FIG. **2**, a block diagram is shown of a symmetrical multiprocessing system **200** including a centralized interrupt controller mechanism. The system **200** includes a plurality of processing units **202-1** through **202-*m*** coupled to a main memory **204** via a CPU local bus **207**. Each processing unit **202-1** through **202-*m*** includes a respective processing core **204-1** through **204-*m***, a respective cache memory **205-1** through **205-*m***, and a respective bus interface **206-1** through **206-*m***. A bus bridge **208** couples CPU local bus **207** to an I/O bus **210**. A plurality of I/O peripheral devices **212-1** through **212-*n*** are coupled to I/O bus **210**. An additional I/O device **214** and an interrupt controller **216** are further coupled to I/O bus **210**. I/O devices **212-1** through **212-*n*** are coupled to a central interrupt control unit **220**. I/O devices **212-1** through **212-*n*** and **214** are accessible by each of the processing units **202-1** through **202-*m*** through the bus bridge **208**.

The central interrupt control unit **220** is provided to manage interrupts received from I/O devices **212-1** through **212-*n*** and interrupt controller **216**, and to distribute the interrupts among the processing units **202-1** through **202-*m***. The central interrupt control unit **220** further manages interprocessor interrupts and software interrupts generated by the processing units **202-1** through **202-*m***. In its preferred form, the central interrupt control unit **220** is implemented with a variety of programmable features as discussed below to accommodate optimal system flexibility.

I/O bus **210** may be any suitable bus for coupling peripheral devices such as CD-ROM units, local area network (LAN) devices, and printers to computer system **200**. Exemplary peripheral bus standards include the ISA (industry standard architecture) bus, the EISA (extended industry standard architecture) bus and the PCI (peripheral component interconnect) bus. Bus bridge **208** provides an interface between I/O bus **210** and CPU local bus **207**.

Processing cores **204-1** through **204-*m*** are data processing units which operate according to a predetermined instruction set. Exemplary processing units include model 80486 processing units, Pentium compatible processing units, and PowerPC processing units. It is understood, however, that processing units **202-1** through **202-*m*** could operate in accordance with still other instruction sets.

Cache memories **205-1** through **205-*m*** are implemented using high speed memory devices. Each cache memory **205-1** through **205-*m*** is associated with a cache controller (not shown separately in the figure) that orchestrates and manages the transfer of data between the associated processing core **204-1** through **204-*m***, the associated cache memory **205-1** through **205-*m***, and CPU local bus **207**. In the preferred form, the cache controller of each processing unit operates concurrently with the associated processing core to provide maximum sustained performance.

CPU local bus **207** has a predetermined bit width and is the computer system's primary bus. Main memory **204** is physical memory of a predetermined size and may be implemented with DRAM (dynamic random access memory). A memory controller (not shown separately) is associated with main memory **204** which controls and orchestrates the transfer of data, address, and control signals communicating between CPU local bus **207** and main memory **204**.

Interrupt controller **216** is provided for sorting and managing interrupt signals derived from a variety of interrupt sources, such as I/O device **214**. Interrupt controller **216** is illustrative of, for example, a model 8259A series programmable interrupt controller manufactured by Advanced Micro Devices, Inc. The 8259A programmable interrupt controller is described within the publication "MOS Microprocessors and Peripherals"; pp. 3–371 through 3–388 Advanced Micro Devices, Inc. 1987).

Before proceeding with a detailed discussion of central interrupt control unit **220**, it is noted at the onset that a variety of configuration registers are embodied within the central interrupt control unit **220**. These configuration registers may be programmed and accessed via I/O bus **210**. Accordingly, the central interrupt controller **220** is independent of the type of CPU local bus **207**, and thus may be employed within a variety of systems using differing types of processing units. As a result, the central interrupt control unit **220** may be used in conjunction with and is compatible with a variety of multiprocessing systems.

Referring next to FIG. **3**, details regarding the central interrupt control unit **220** are next considered. FIG. **3** is a block diagram of one embodiment of central control unit **220** including a central controller **302** coupled to an I/O interrupt controller **304** and to a processor interrupt generator **306**. An I/O bus interface unit **308** is further shown coupled to central controller **302**.

I/O bus interface unit **308** provides an interface between I/O bus **210** and central controller **302** to allow program-

ming of the central control unit **220**, as well as to accommodate other functions of the central control unit **220**, as will be explained in greater detail below. It is understood that the central interrupt control unit **220** could alternatively or additionally include a CPU bus interface unit for coupling the CPU local bus **206** to the central control unit **220** for programming and other functions.

As explained previously, the central interrupt control unit **220** is capable of accepting interrupts from a variety of different I/O devices. These interrupts are received at a plurality of interrupt pins, labeled INTR1-INTRn, and are provided to I/O interrupt controller **304**. The central interrupt control unit **220** is configured such that each interrupt INTR1-INTRn can be individually programmed to designate a specific type of interrupt, to specify a particular delivery mode, and to indicate its priority level. In addition, each interrupt pin can be utilized in a cascaded mode to expand the number of interrupt signals which can be received and identified at the particular pin. This will be explained further below.

Central controller **302** prioritizes the various interrupt signals and routes them to the processor interrupt generator **306**, which responsively routes the interrupt signals to one or more of the processing units **202-1** through **202-*m*** based on the delivery mode for each interrupt and the current task priority of each processing unit, among other things. The central controller **302** maintains an interrupt stack and a device table for the system, and further maintains the current task priorities of all processing units. The central controller **302** finally includes a mechanism for distributing selected interrupts that need to be handled by all processors in a broadcast fashion. This distribution mechanism will be explained in greater detail below.

As stated previously, processor interrupt generator **306** routes the various interrupts to a designated destination processing unit (or processing units). In this embodiment, the central interrupt control unit **220** is configured to distribute interrupts among a maximum of **256** processing units. The number of processing units provided within the system is programmed upon system initialization, as will be understood from the description below.

FIG. **4** is a block diagram of I/O interrupt controller **304**. The I/O interrupt controller **304** receives interrupts from I/O devices via pins INTR1, INTR2, . . . INTRn. The I/O interrupt controller **304** includes a plurality of interrupt channels **402-1** through **402-*n*** coupled to pins INTR1-INTRn, respectively. A central controller interface **404** is coupled to each to interrupt channel **402-1** through **402-*n***. The interrupt channels **402-1** through **402-*n*** provide dedicated channels through which interrupts received at the associated interrupt pins INTR1-INTRn, respectively, are processed. In one embodiment, the I/O interrupt controller **304** includes a total of sixteen interrupt input pins, each of which can be cascaded with sixteen interrupt signals to support a maximum of 256 unique interrupt vectors.

A plurality of registers (not shown in FIG. **4**) are provided within each interrupt channel **402-1** through **402-*n*** to control the processing of each incoming interrupt. These registers are mapped either within the memory space or the I/O space of the system. Further details regarding the internal registers of interrupt channels **402-1** through **402-*n*** are provided below.

Each interrupt channel **402-1** through **402-*n*** detects the assertion of an interrupt signal at its associated input pin INTR1-INTRn and processes the interrupt signal to verify whether the interrupt should be dispatched to the processing units. Interrupt characteristics can be programmed individually, and implied positional dependence is not attached to any of the interrupt channels **402-1** through **402-*n***.

FIG. **5** is a block diagram illustrative of each of the interrupt channels **402-1** through **402-*n***. The interrupt channel **402** of FIG. **5** includes an interrupt input processor **502** coupled to a registers unit **504** and an interrupt acceptance unit **506**. The interrupt input processor **502** processes the interrupt signal (or signals, if cascade mode is programmed) on the INTR pin and determines the mode of transfer for cascaded interrupts. If the pin is programmed to be a cascaded pin, the index of the current cascaded interrupt is determined and stored in a cascade interrupt address (CIA) register. It is noted that when the interrupt channel **402** is operated in a cascade mode, the number of registers units **504** and the number of interrupt acceptance units **506** are effectively duplicated fifteen times, such that a separate interrupt sub-channel is provided for each possible cascaded interrupt signal. These sub-channels are illustrated in phantom within FIG. **5**.

Each interrupt signal is associated with a programmable control register **504A**, a destination CPU register **504B**, an affinity CPU register **504C**, and an ID (vector) register **504D**. Based on the information within the control register **504A**, the interrupt acceptance unit **506** processes the signal on the INTR pin. If the interrupt is a genuine, enabled, and acceptable signal, it is passed on the central controller interface **404** (of FIG. **4**) to be delivered to one or more of the processing units.

As stated previously, each INTR pin can be programmed in a cascade mode wherein the interrupt pin may receive a cascaded signal representing sixteen distinct interrupts. When a pin is programmed to be cascaded, sixteen individual sets of registers units and interrupt acceptance units will be associated with the interrupt channel, as illustrated in phantom in FIG. **5**. The cascade modes of central control unit **220** will be explained in greater detail below.

The various registers of each interrupt channel (or sub-channel) are next considered. As stated previously, a separate set of registers are provided for each possible interrupt signal. These registers are labeled the control register **504A**, the destination CPU register **504B**, the affinity CPU register **504C**, and the ID register **504D**. These registers are visible to software and are located in either memory mapped or I/O mapped system space. A separate control register **504A** is provided for each interrupt signal and is programmable. The control register **504A** for each interrupt signal defines and dictates the functionality of each INTR pin, and each may be embodied by a 32-bit register. FIG. **6** illustrates the fields associated with the control register **504A**, and Table 1 describes the various fields of the control register **504A**. Table 2 indicates the cascade mode encoding, Table 3 indicates the delivery mode encoding, and Table 4 indicates the status bits encoding. As indicated by Tables 1–4, the control register **504A** stores various information for defining the type of interrupt signal that will be provided to the channel, the mode of the interrupt pin (i.e., normal or cascade mode), whether the interrupt signal is currently masked, the priority level associated with the interrupt, the delivery mode, along with other parameters associated with the interrupt.

TABLE 1

IIC Control Register Fields

Field	Name	Bits	Explanation
TT	Trigger Type	1	Interrupt is edge triggered or level triggered
IP	Interrupt Polarity	1	Polarity of the interrupt (active high or low)
EN	Enable	1	Corresponding interrupt is being used
CSD	Cascade	1	Enables the pin to be a cascaded interrupt
MSK	Mask	1	Interrupt is masked
CM	Cascade Mode	2	Cascaded Interrupt vector delivery mode - refer to Table 2
PL	Priority Level	4	Priority Level assigned to the interrupt
DM	Delivery Mode	3	Mode of delivering the interrupt - refer to Table 2
AC	Affinity CPU	1	This interrupt has an affinity CPU
ST	Status	2	Current status of the interrupt - refer to Table 4
EOI	EOI enabled	1	Enable EOI latency timer for level triggered interrupts
ISA	ISA mode	2	System is in ISA mode

TABLE 2

Cascade Mode Definition

CM(1:0)	Definition
00	Serial coded mode
01	Serial 16-bit mode
10	8259 mode
11	TBD

TABLE 3

Delivery Mode Definition

DM(2:0)	Definition
000	Fixed: Deliver the interrupt to the CPU/CPUs in destination CPU register
001	Lowest Priority: Delivery the interrupt to the processor executing at lowest priority
010	Broadcast: Deliver the interrupt to all the CPUs
011	TBD
100	NMI: Deliver a level triggered interrupt to destination CPUs as NMI
101	Reset
110	SMI: Deliver a level triggered interrupt to destination CPUs as SMI
111	TBD

TABLE 4

Status Bits Definition

ST(1:0)	Definition
00	Idle
01	Being serviced (dispatched and acknowledged)
10	Dispatched from the CIC (but not acknowledged)
11	Queued in the CC

Referring back to FIG. **5**, the definition of the destination CPU register **504B** depends on the delivery mode and the current status of the associated interrupt signal. If the interrupt is not being serviced, the destination CPU register **504B** has the ID of the processing unit or group of processing units that the interrupt is being targeted to. If the interrupt is being serviced, the destination CPU register **504B** has the ID of the processing unit that is servicing the interrupt. If the delivery mode is broadcast or lowest priority, this register conveys no associated meaning.

The affinity CPU register **504C** holds the ID of the processing unit **202-1** through **202-*m*** (FIG. **2**) that serviced the interrupt most recently. The ID register **504D** contains the ID (or vector) of the interrupt.

Interrupts are processed by the respective interrupt acceptance unit **506** before passing them on to the central controller **302** via central controller interface **404**. If the interrupt is enabled (EN of ICR) and not masked (MSK of ICR), it is passed on to the central controller **302** along with the information about the delivery mode, destination processing unit (if any), priority level and interrupt ID.

As stated above, the architecture allows for each interrupt pin INTR1-INTRn of the central interrupt control unit **220** to be programmed as either a direct interrupt or as a cascaded interrupt. If the cascade bit in the control register **504A** is set, the associated interrupt channel accommodates a total of fifteen additional expansion interrupts. Each of these expansion interrupts are associated with a dedicated interrupt control register (ICR) **504A**, a destination CPU register **504B**, an affinity CPU register **504C**, and an ID register **504D**, as illustrated in phantom in FIG. **5**. Except for the fact that the CSD, CM and ISA fields of the expansion interrupt control registers are undefined, these registers are identical to the ones defined by Tables 1–4.

The CM field determines the method used to access one of sixteen interrupts in cascade mode. The index of one of sixteen interrupts is determined by the CIA register located in the interrupt input processor. The CM mode determines the mechanism of computing the CIA.

FIGS. **7** and **8** illustrate hardware configurations for cascading a particular pin. It is noted that the central interrupt control unit **220** supports interrupt expansion via three different cascading modes. The first two modes allow a single physical interrupt pin INTR1-INTRn to route the interrupts of multiple I/O devices to selected interrupt subchannels. The third mode allows for the integration of a conventional 8259-type interrupt controller within the computer system. This provides traditional PC hardware and software compatibility.

Referring first to FIG. **7**, a hardware configuration is illustrated for what is referred to as the "serial coded" mode. FIG. **7** is a block diagram that illustrates internal portions of the interrupt input processor **502** which are activated when the particular interrupt channel is set in the serial coded cascade mode. As illustrated in FIG. **6**, the interrupt input processor **502** includes a control unit **570** coupled to a shifter control **572** and a serial decode circuit **574**. The control unit **570** is responsive to the CSD and CM fields of the corresponding control register **504A**. When the CM field (i.e., cascade mode field) indicates that the current mode is serial coded cascade mode, the control unit **570** activates the shifter control **572** and the serial decode circuit **574** such that serially transmitted encoded data at the corresponding interrupt pin INTR is decoded to identify activity of a particular interrupt signal. The interrupt input processor **502** is shown coupled to a remote interrupt handler **580** which includes an interrupt data encoder coupled to a shift register **584** and to a parallel interrupt detect circuit **586**.

A plurality of interrupt signals 0 to 15 are provided to parallel interrupt detect circuit **586**. The remote interrupt handler **580** gathers the device interrupts and communicates the status of each interrupt signal to the central interrupt control unit via a coded serial message on the INTR line of the associated interrupt channel. Parallel interrupt detect circuit **586** monitors the interrupt signals provided to the remote interrupt handler **580**. If a transition occurs in any of the interrupt signals, the interrupt data encoder **582** causes an encoded serial message to be broadcast to the interrupt input processor **502** via shift register **584**. The serial data is transmitted on a serial coded message line which is provided to the INTR line of the interrupt channel. The shift register **584** provides a synchronizing shift clock to the interrupt input processor **502** as each serial coded message is being transmitted.

In one embodiment, the encoded messages consist of a cascaded interrupt signal number and interrupt state in a 6-bit form. A possible encoding scheme is as follows:

Bits 5:4 Interrupt State	00	interrupt transitioned to low
	01	interrupt transitioned to high
	10	reserved
	11	reserved
Bits 3:0 interrupt number for cascaded interrupt input signals numbered 0 to 15		

Consider, for example, a situation in which interrupt signal **3** transitions from low to high. This transition is detected by the parallel interrupt detect circuit **586**. The interrupt data encoder **582** responsively generates an encoded value indicative of the type of transition that occurred and the particular interrupt signal that made the transition. For example, if the above coding scheme is employed, an encoded value of "010111" would represent a transition to high ("01") in interrupt signal **7** ("0111"). The encoded value is then provided to shift register **584**, which initiates a serial transmission which is received by the shifter control unit **572**. The serial decode unit **574** then decodes the received message in accordance with the coding scheme of the interrupt data encoder **582**, and provides the message to the interrupt acceptance unit **506** of the designated interrupt sub-channel. The interrupt acceptance unit **506** for the corresponding interrupt signal then passes the interrupt on to the central controller **302** if the interrupt is enabled (EN of the control register) and not masked (MSK of control register). Similar to the previous description, when an interrupt acceptance unit **506** passes an interrupt on to the central controller **302**, the interrupt is passed along with the information regarding the delivery mode, the destination CPU (if any), the priority level, and the interrupt ID for the interrupt signal.

In accordance with the serial coded cascade configuration of FIG. **6**, the serial channel is active only when activity occurs on one or more of the interrupt signal lines, and thus provides a low power and electrically-quiet expansion technique. Furthermore, the encoding scheme provides reserved values to accommodate additional types of messages.

FIG. **8** is a block diagram of an alternative cascading configuration. Circuit portions that correspond to those of FIG. **7** are numbered identically. In this configuration, rather than encoding a value indicative of activity of a particular interrupt signal, the state of the parallel interrupt detect circuit **586** is communicated continuously and directly to interrupt input processor **502** via shift register **584**. As such, shift register **584** continuously generates a serial signal indicative of the state of parallel interrupt detect circuit **586**, and shifter control unit **572** converts the serial transmission to parallel data. The parallel data is then decoded by message decoder **590** which passes a detected interrupt signal transmission to the corresponding interrupt acceptance unit **506** along with the associated control and vector information within the corresponding registers unit **504**.

In this configuration, the interrupt input processor **502** continually clocks the remote interrupt shift register **584**. It must then keep track of which interrupt the current data belongs to and route it to the appropriate channel. The data is simply "interrupt high" or "interrupt low". The central controller (or interrupt acceptance unit **506**) must then determine if the data represents a change in the interrupt state and therefore what action, if any, should be taken.

Additional aspects of the central interrupt control unit **220** of FIG. **2** are considered next. Referring to FIG. **9**, a block diagram is shown which is illustrative of the processor interrupt generator **306**. As stated previously, the processor interrupt generator **306** receives interrupt information from the central controller **302** and generates processor interrupt signals labeled INT1-INTm to be delivered to the processing units. As illustrated in the figure, the processor interrupt generator **306** includes a central controller interface **602**, an interprocessor interrupt (IPI) and software interrupt register set **604**, and a set of CPU channels **606-1** through **606-*m***. Each processing unit in the system receives an interrupt from an associated CPU channel **606-1** through **606-*m*** of the processor interrupt generator **306**. The CPU channels **606-1** through **606-*m*** receive interrupts from the central controller **302** (FIG. **3**) through central controller interface **602** and dispatches them to the appropriate processing unit (or units).

FIG. **10** is a block diagram illustrative of each of the processor channels **606-1** through **606-*m***. The CPU channel **606** of FIG. **10** includes a CPU channel registers unit **650** and an interrupt queue **652** coupled to an interrupt dispatch control unit **654**. The interrupt dispatch control unit **654** dispatches pending interrupts to the corresponding processing unit.

The CPU channel registers unit **650** includes a current task priority register **650A**, a current interrupt ID register **650B**, a processor ID register **650C**, and a control register **650D**. The functions and bit definitions of each of these registers is described next.

Each processing unit in the system is assigned a dedicated control register **650D** to dictate the functionality as seen by the central interrupt control unit **220**. These are 32-bit programmable registers which are mapped within either I/O or memory space of the system. FIG. **10A** illustrates a CPU channel control (CIG) register **650D** along with its associated fields, and Tables 5 through 7 describe each of the fields within the control register.

TABLE 5

CIG Control Register Fields

Field	Name	Bits	Explanation
EN	Enable	1	This CPU channel is being used
IEN	Interrupt Enable	1	The CPU connected to this channel will accept the interrupts
ITM	Interrupt Transmit Mode	2	Mechanisms for delivering the interrupts to the CPU - refer to Table 6

TABLE 5-continued

CIG Control Register Fields

Field	Name	Bits	Explanation
RMI	Real Mode Interrupt	1	This interrupt is delivered in "real mode"
IML	Interrupt Mask Level	4	Mask all the interrupts at or below this priority level
IST	Interrupt Status	2	Reflects the status of the interrupt on this channel - refer to Table 7

TABLE 6

Ioterrupt Transmission Mechanism Definition

ITM(1:0)	Definitioo
00	Deliver the interrupts on separate pins
01	Deliver the interrupts on the CPU bus
10	Use dedicated interrupt delivery bus (say, 4-bit wide) to deliver the interrupts
11	TBD

TABLE 7

Interrupt Status Definition

IST(1:0)	Definition
00	No interrupt is being serviced
01	Interrupt being serviced by the CPU
10	Interrupt dispatched to the CPU, not yet acknowledged
11	TBD

The processor ID register **650C** contains the ID of the processing unit **202-1** through **202-*m*** associated with the specific channel. The current interrupt ID register **650B** is provided to store the ID (vector) of the interrupt that is being serviced by the processing unit connected to thc channel. The current interrupt ID register **650B** is valid only when the status field of the control register indicates that an interrupt is being serviced. The current task priority register **650A** reflects the priority of the task being executed by the processing unit affiliated with the channel.

Referring back to FIG. **9**, the interprocessor interrupt and software interrupt register set **604** provides a set of registers logically accessible at the same locations from all of the CPU channels. The space in this register set provides unique register views to each CPU channel by using the processor ID as an index. Thus, when two processors generate read/write cycles to these registers mapped at the same logic location, they will actually be accessing separate physical registers. The processing units write to these registers to initiate interprocessor interrupts or to schedule software interrupts.

FIG. **10B** illustrates the interprocessor interrupt (IPI) register format and its fields. All IPI registers are accessible to the software at either an I/O location or a memory location of the system. The ID of the processing unit is used as an index to determine which register is being accessed. Tables 8 through 11 provide descriptions of the various fields within each IPI register.

TABLE 8

CIG IPI Register Fields

Field	Name	Bits	Explanation
TT	Trigger Type	1	Interrupt is edge triggered or level triggered
IP	Interrupt Polarity	1	Polarity of the interrupt (active high or low)
EN	Enable	1	Corresponding interrupt is being used
ST	Status	2	Curreot Status of the IPI - refer to Table 9
DTC	Destination Code	2	IPI's destination code - refer to Table 10
PL	Priority Level	4	Priority Level assigned to the interrupt
DM	Delivery Mode	3	Mode of delivering the interrupt - refer to Table 11
DID	Destination ID	8	Destination processor ID to which the IPI is scbeduled for
SID	Source ID	8	Originatiog processor ID

TABLE 9

IPI Register Status Field Definitioo

ST(1:0)	Definition
00	Idle
01	Beiog serviced (dispatched and acknowledged)
10	Dispatched from the CIC (but oot acknowledged)
11	Queued in the CC

TABLE 10

IPI Register Destination Code Field Definitioo

DTC(1:0)	Definition
00	Destination ID field (DID) of the IPI register
01	Self
10	Broadcast
11	All processor excludiog self

TABLE 11

IPI Register Delivery Mode Field Definition

DM(2:0)	Definition
000	Fixed: Deliver the interrupt to the CPU/CPUs io destination CPU register
001	Lowest Priority: Deliver the interrupt io the processor executing at lowest priority
010	TBD
011	TBD
100	NMI: Deliver a level triggered interrupt to all CPUs as NMI
101	Reset:
110	TBD
111	TBD

A processing unit performs a write to its IPI register when it has scheduled an interprocessor interrupt. If a processing unit can schedule multiple interprocessor interrupts, it should monitor the ST (status) field of the IPI register. If this field is idle, then the processing unit can inject an interprocessor interrupt into the system. If a processing unit injects an interprocessor interrupt without checking the status of the IPI register and if the ST field is not idle, then the current

interprocessor interrupt and any previously scheduled interprocessor interrupts destiny will be undetermined. It is noted that a software interrupt register for each processor channel may further be provided which has a format identical to that specified for the interprocessor interrupt register. For software interrupts, however, a requested interrupt is delivered only to the interrupt-requesting processing unit.

Referring back to FIG. **2**, details regarding the start-up of the computer system **200** as well as the initialization of the various configuration registers within central interrupt control unit **220** are next considered. During system configuration, one of the processing units **202-1** through **202-*m*** is designated as the "boot" processing unit. It is assumed in the below discussion that processing unit **202-1** has been designated as the boot processing unit. FIG. **11** is a flow diagram which illustrates the operation of the boot processing unit upon system reset. During a step **852**, the processing unit **202-1** begins a power-on self test procedure and an initialization procedure. It is noted that initially, the other processing units **202-2** through **202-*m*** are held in reset by central control unit **220**. During step **854**, the processing unit **202-1** initializes the registers unit **504** (i.e., the control register **504A**, and the ID (vector) register **504D**) of each interrupt channel. As stated previously, the registers unit **504** of each interrupt channel is mapped within the I/O or memory space of the computer system. Each register of each interrupt channel is designated with a predetermined and unique address. The initialization data provided to the registers unit **504** of each interrupt channel is typically stored within the BIOS code of main memory **204**. As such, the BIOS code for initializing the registers unit **504** of each interrupt channel is dependent upon the particular system configuration (i.e., the number and type of interrupt-generating resources) and must be provided by the system programmer.

The CPU channel **606-1** of central interrupt control unit **220** that connects to processing unit **202-1** must also be initialized. It is noted, however, that the current task priority register **650A**, the current interrupt ID register **650B**, the processor ID register **650C**, and the control register **650D** for a particular CPU channel **606-1** through **606-*m*** reside and are mapped at the same system address locations (either I/O or memory space) as the corresponding registers for the other CPU channels. That is, the address of the current task priority register **650A** is identical for each CPU channel **606-1** through **606-*m***. Similarly, the address of current interrupt ID register **650B** for each CPU channel is identical, as are the address values for the processor ID register **650C** and the control register **650D** of each CPU channel. A processing unit ID value is thus associated with each processing unit **202-1** through **202-*m*** which is embedded within a designated command for initializing or updating the CPU channel registers unit **650** of each CPU channel **606-1** through **606-*m***. This will be explained in greater detail below.

FIG. **12** is a diagram that illustrates in greater detail the hardware which enables the CPU channel registers unit **650** of each CPU channel **606-1** through **606-*m*** to be initialized and updated during normal execution. FIG. **12** illustrates an ID register **902-1** which is associated with processing unit **202-1**. Identical ID registers **902-2** through **902-*m*** are also associated with processing units **202-2** through **202-*m***, respectively. Each ID register **202-1** through **202-*m*** contains a value which uniquely identifies the particular processing unit. The ID value of each processing unit may be a hardwired value or may be provided during system configuration. For example, if fifteen processing units are connected within the system, the ID values within ID registers **902-1** through **902-16** may range from 0 to 15, respectively. Each ID register **902-1** through **902-*m*** may be accessed through software command via associated control decoders **904-1** through **904-*m*** of each processing unit. The ID registers may be mapped within either memory or I/O space. It is noted, however, that each processing unit accesses its corresponding ID register **902** via the same address value. For example, the ID register **902** of each processing unit **202-1** through **202-*m*** may be mapped at a memory location of 2000:H. Thus, if a designated processing core **204-1** through **204-*m*** executes a read cycle to the memory location 2000:H, the value residing within the corresponding ID register **902** for that processing unit will be provided to the processing core. Each processing core would read a unique value in these situations.

FIG. **13** illustrates circuitry embodied within the central interrupt control unit **220** that allows data to be written (or read) from the respective CPU channel registers unit **650** of each CPU channel **606-1** through **606-*m***. In this illustration, it is assumed that a total of sixteen processing units may be connected within the system; however, it is understood that the circuitry may alternatively be configured to accommodate, for example, 256 unique processing units. FIG. **13** illustrates the CPU channel registers units **650-1** through **650-16** for the separate sixteen CPU channels. As stated previously, each CPU channel registers unit **650-1** through **650-16** includes a current task priority register **650A**, a current interrupt ID register **650B**, a processor ID register **650C**, and a control register **650D**. Each of these registers is coupled to receive (or provide) data from the data lines of CPU local bus **207**. A 4-to-16 decoder circuit **920** is further coupled at its inputs to selected data lines of CPU local bus **207**. The outputs of the 4-to-16 decoder circuit **920** are coupled to respective select lines of the CPU channel registers units **650-1** through **650-16**. It is noted that each CPU channel registers unit **650-1** through **650-16** receives a separate select signal from 4-to-16 decoder circuit **920**. An address decoder **922** is further coupled at its input to the address lines of I/O bus **210**. Four latch enable lines are provided at an output of address decoder **922**. An address decoder enable line is coupled to the latching enable inputs of each current task priority register **650A** of the CPU channel registers units **650-1** through **650-16**, and similarly address decoder enable lines are further connected to each current ID register **650B**, each processor ID register **650C**, and each control register **650D**.

In accordance with the hardware implementation as depicted by FIGS. **12** and **13**, when the CPU channel registers unit **650** of a designated CPU channel **606-1** through **606-*m*** must be initialized or updated, the operating system programmer may structure the executing code such that the processor ID value within a designated ID register **902** is embedded as an index to direct attached data to the correct CPU channel registers units **650**. For example, consider a situation in which the ID register **902** of each processing unit **202-1** through **202-*m*** is mapped at a memory location 2000:H, and wherein the control register **650D** of each CPU channel **606-1** through **606-*m*** is mapped at an I/O address of 3000:H. If the operating system must update the configuration information within control register **650D** for a particular processing unit, the programmer may first cause the designated processing unit to execute a memory read cycle to memory location 2000:H to read the value within the particular processing unit's ID register. The programmer may then invoke a command to append the ID value with the configuration data to be stored within the associated control

register **650D**. Subsequently, an I/O write command to address location 3000:H is executed to write the combined information (i.e., the configuration data along with the processor ID value). This I/O cycle is decoded by address decoder **922** which responsively causes the control registers **650D** of each CPU channel registers unit **650-1** through **650-16** to be enabled. The processor ID value which is appended with the configuration data is then decoded by 4-to-16 decoder **920**, which provides a select signal to a selected one of the CPU channel registers units **650-1** through **650-16**. This causes the configuration data to be stored within only the selected and enabled registers. Configuration data is thereby provided to the designated CPU channel **606-1** through **606-*m*** without requiring separate, dedicated address locations for the configuration registers of each CPU channel. It is noted that cycles for updating the registers of each CPU channel as well as read cycles are accomplished similarly. Exemplary code that carries out the required processor ID read operation as well as the code to append the ID to the configuration data and to write the configuration data to a designated CPU channel is as follows:

```
OS Code to Init a given CPU's control reg
    procedure init_channel
        id:= get mem(2000); - read ID register
        command:= id<< 28 or config data
        put io(3000), command
        end
```

Referring back to FIG. **11**, after the boot processing unit has initialized the I/O channels in the central interrupt control unit **220** (the configuration registers of each I/O channel are mapped at dedicated locations separately from the configuration registers from the other I/O channels), the boot processing unit **202-1** must initialize the CPU channel **606-1**. This is accomplished using the method described above in conjunction with FIGS. **12** and **13**. Thus, during step **856**, the processing unit **202-1** reads its corresponding ID register **902**. During step **858**, the processing unit **202** appends its ID register value to the desired configuration data which must be stored within a designated register of CPU channel registers unit **650**. The processing unit **202-1** then executes a cycle to write the combined data to the selected register of CPU channel registers unit **650**. It is noted that during this cycle, the 4-to-16 decoder **920** of FIG. **13** is employed to select the registers unit **650-1** of the CPU channel **906-1**. Similar operations may be initiated to write additional initialization data into other registers of the CPU channel registers unit **650** of CPU channel **606-1**. For one implementation, the current task priority register **650A** and the control register **650D** are written with initialization data by processing unit **202-1** during the initialization sequence. After the processing unit **202-1** has initialized its CPU channel **606-1**, the processing unit **202-1** provides a command to the central interrupt control unit **220** which causes the central interrupt control unit **220** to release the remaining processing units **202-2** through **202-*m*** from reset (step **862**). Subsequently (step **864**), the boot processing unit **202-1** waits for the slave initialization sequence as depicted in FIG. **14** to complete. This completes the initialization sequence.

FIG. **14** is a flow diagram that depicts the initialization sequence of each slave processing unit **202-2** through **202-*m***. When the master processing unit **202-1** causes the central interrupt control unit **220** to release the remaining processing units from reset, each processing unit **202-2** through **202-*m*** reads its associated ID register **902-2** through **902-*m*** respectively during step **870**, appends the ID value with the configuration data to be stored within the control register **650D** (step **872**), and writes the combined data to the CPU channel registers unit **650** (step **874**). The decoder circuit **920** of FIG. **13** is active during these cycles to select the appropriate CPU channel registers unit **650-1** through **650-16** in accordance with the processor ID identified during each particular cycle. Similar operations are repeated to initialize the task priority register **650A** of each CPU channel (step **876**). It is noted again that since a unique ID value residing within each of the ID registers **902-2** through **902-*m*** is appended to the data written to each register of CPU channel registers unit **650**, each processing unit **202-2** through **202-*m*** effectuates its own CPU channel configuration.

Referring back again to FIGS. **2** and **4**, it was stated previously that each designated interrupt channel **402-1** through **402-*n*** may be programmed in what is referred to as the "8259" mode. This allows a programmable interrupt controller such as interrupt controller **216** to be connected to central interrupt control unit **220** when a particular interrupt channel is programmed in the 8259 mode (as indicated by the CM field of the associated control register **504A**). During the 8259 mode of operation, the interrupt signal from the 8259 interrupt controller is passed through the central interrupt control unit **220** in accordance with its programmed priority, and the acknowledge signal from the receiving processing unit **202-1** through **202-*m*** is passed back through the central interrupt control unit **220** to the interrupt controller **216**. This is depicted within FIG. **15** which shows one of the processing units **202** coupled to the central interrupt control unit **220** and to a buffer **219**. When an ISA interrupt is received at an interrupt input of the 8259 interrupt controller **216**, the interrupts are passed through the central interrupt control unit **220** and, in accordance with the programmed priority level and other central control routing, is passed to a designated processing unit **202**. When the designated processing unit **202** acknowledges the cycle, the interrupt acknowledge signal INTA is passed through the central interrupt control unit **220** and is received at the interrupt acknowledge line of the 8259 interrupt controller **216**. The interrupt controller **216** responsively drives the interrupt vector on an X-bus **211** (or any other bus), and the interrupt vector is passed to the processing unit **202** via a buffer **219**. It is noted that buffer **219** may be embedded within bus bridge **208**. Accordingly, for the 8259 cascade mode, the central interrupt control unit **220** does not directly respond to the interrupt acknowledge cycle of the receiving processing unit **202**, and instead allows the vector information to be provided from the 8259 interrupt controller **216**. It is further noted that accommodation of the 8259 cascading mode as described above advantageously allows the use of integrated interrupt sources such as a model 82C206 integrated circuit which includes a system timer **834** and a real time clock **835**.

Referring back to FIG. **2**, the multiprocessing system **200** may further be configured to allow the transfer of interrupt information across various interfaces using a specially defined cycle which is transferred across one or more of the buses incorporated within the system. This will be best understood with reference to FIG. **16**. FIG. **16** is a block diagram of the multiprocessing system as generally represented in FIG. **2** with an additional I/O device **280** coupled to a second I/O bus **282**. The second I/O bus **282** is coupled to I/O bus **210** via a bus bridge **284**. The bus bridge **284** is illustrative of, for example, a docking station for coupling a portable computer as represented by I/O device **280** to the

multiprocessing system. For the system of FIG. **16**, the I/O device **280** may provide an interrupt signal to bus bridge **284**. However, due to the cost and the possible unavailability of dedicated interrupt pins that coupled bus bridge **284** to central interrupt control unit **220**, bus bridge **284** may not be configured to assert an interrupt signal at a dedicated line that is received by central interrupt control unit **220**. Instead, in response to the assertion of an interrupt by I/O device **280**, the bus bridge **284** may effectuate a specialized cycle, or a memory or I/O cycle to a dedicated memory location, to which a particular interrupt channel of central interrupt control unit **220** will respond. FIG. **17** illustrates an interrupt channel including hardware configured to decode an interrupt cycle as executed by bus bridge **284** and to assert a corresponding interrupt signal within the corresponding interrupt channel. As illustrated in FIG. **17**, a control unit **290** is responsive to the CM field of control register **504A** and correspondingly enables the interrupt cycle decoder **292** if the interrupt channel mode is designated as an I/O bus mode. When control unit **290** enables interrupt cycle decoder **292**, the specialized interrupt cycle generated by bus bridge **284** is detected by interrupt cycle decoder **292**, which correspondingly asserts an interrupt signal at the INTR input of interrupt input processor **502**. It is noted that for situations in which I/O bus **210** is a PCI standard configuration bus, the special interrupt cycle may be defined by pre-specified coding of the cycle definition bits of the PCI bus. Alternatively, the special interrupt cycle may be defined as a cycle to a predetermined address in either the memory or I/O space of the system.

The prioritization of interrupts by central controller **302** is next considered. FIG. **18** is a block diagram that depicts a portion of central interrupt control unit **220** including I/O interrupt controller **304**, central controller **302**, and processor interrupt generator **306**. Rather than automatically associating a particular interrupt with a fixed priority depending upon its interrupt vector, the central interrupt control unit **220** allows each interrupt to have a separate programmable interrupt vector and a separate priority. The priority is stored within the PL field of the associated control register **504A** for the interrupt channel. As stated previously, upon system initialization, the vector for each interrupt channel is set. In addition, the priority level for the interrupt channel is also set. Once a particular interrupt request is accepted by the I/O interrupt controller **304**, the interrupt vector and priority data is processed by an interrupt scheduler **305** of the central controller **302** which correspondingly provides the interrupt vector and priority data for each interrupt within a pending interrupt queue **652** (FIG. **10**) of a designated CPU channel of the processor interrupt generator **306**. The interrupts are provided to the various interrupt queues of the CPU channels in a prioritized manner based upon the priority level indicated by the interrupts control register as well as based upon the current task priorities of the available processing units. FIG. **18** illustrates a decoder unit **309** within the I/O bus interface unit **308** which allows separate programming of the vector information and priority information for each interrupt via designated I/O or memory cycles on I/O bus **210**. For one simple configuration, the pending interrupt requests are provided to the interrupt queues of the CPU channels in their prioritized order.

Referring again to FIG. **2**, it was stated previously that selected interrupts such as a timer tick interrupt may need to be provided in a broadcast fashion to each processing unit **202-1** through **202-*m***. If the delivery mode for a particular interrupt signal is designated as broadcast mode (i.e., DM field of the control register **504A** for the interrupt), the central controller **302** operates in accordance with an auto-chaining technique as depicted in the flow diagram of FIG. **19**. As illustrated in FIG. **19**, if the interrupt request signal for an interrupt designated as broadcast is asserted as determined during step **470**, the interrupt is provided to the processing unit **202-1** through **202-*m*** that has the lowest current task priority level (as indicated by the current task priority register **650A** for that CPU channel) during step **472**. During step **474**, the designated processing unit services the interrupt and returns an End Of Interrupt (EOI) command to the central controller **302**. If all processing units **202-1** through **202-*m*** have not yet received the interrupt (step **476**), the interrupt is provided to the next processing unit **202-1** through **202-*m*** which has not yet received this interrupt and which has the lowest current task priority value (of those remaining processing units that have not yet received the interrupt). This next processing unit then services the interrupt and returns an End Of Interrupt command to the central controller **302**. This process repeats until each processing unit **202-1** through **202-*m*** has serviced the interrupt. When all processing units have received and serviced the interrupt, the busy bit for the interrupt (i.e., the idle state of the ST field of the control register **504A** for the interrupt channel) is cleared (step **478**). It is understood that a similar auto-chaining procedure may be carried out if a designated set of processing units must receive a particular interrupt, rather than all the processing units. In accordance with the auto-chaining technique described above in which interrupts such as timer tick interrupts are provided to two or more of the processing units, the central controller **302** intelligently selectively passes the interrupt to the processing unit having the lowest current task priority level indicated. The interrupt is not passed to subsequent processing units until the prior processing unit has completed its service of the interrupt. As a result, bus contention is minimized and system performance is maximized by interrupting the processing units with the lowest relative current task priority values before interrupting processing units with high relative task priorities.

The central interrupt control unit **220** is further configured to prevent the occurrence of spurious interrupts. As stated previously, when a level triggered interrupt is being serviced by a designated processing unit, an I/O command is typically associated with the interrupt service routine that, when executed, causes the interrupt source to deassert the interrupt signal. Subsequently, an End Of Interrupt (EOI) command is executed to inform the central interrupt control unit **220** that the interrupt service has completed. A spurious interrupt may occur if a significant latency is introduced between the time at which the processing unit executes the I/O command (to cause the interrupt source to deassert the interrupt signal) and the time at which the interrupt signal is deasserted. Such a latency may occur, for example, if the I/O device resides on a remote bus via several bus interface units. If the interrupt source does not deassert the interrupt signal before the centralized interrupt controller responds to the End Of Interrupt command, the continued assertion of the interrupt may be detected by the central interrupt control unit **220**, thus, causing the interrupt to be unintentionally re-initiated.

Referring to FIG. **20**, the central control unit **220** is advantageously associated with a programmable latency timer **595** coupled to I/O bus **210** through a decoder unit **596**. The programmable latency timer **595** may be programmed by a system user to set a programmable time delay between the time at which the central controller **302** receives an End Of Interrupt command and the time at which the central controller **302** resets the status (ST) field of the control

register **504**A of the particular interrupt channel. It will be appreciated that the I/O interrupt controller **304** is configured such that a designated interrupt signal is not monitored if the status indicates that the interrupt is either being serviced, has been dispatched from the central interface control unit **220** to a particular processing unit, or has been queued in the central controller **302**. Once the central controller **302** resets the status to idle for a particular interrupt signal, the I/O interrupt controller **304** continues to monitor the particular interrupt signal for subsequent assertions. It is noted that since programmable timer **595** delays the resetting of the ST field of the control register for the particular interrupt channel, spurious interrupts may be prevented. It is further noted that the programmable timer **595** may be selectively enabled for each interrupt via the EOI field of that interrupt's channel control register **504**A. It is also noted that decoder **596** may be embodied as an integral part of I/O bus interface unit **308**.

Referring finally to FIGS. **21** and **22**, the integration of system management interrupts into the symmetrical multiprocessing system of FIG. **2** is next considered. As is generally known, system management interrupts are commonly used, for example, in the implementation of system power management. However, for a symmetrical multiprocessing system, SMI interrupts typically cannot be handled in the same way as normal interrupts and NMI (non-maskable interrupts) are handled. Accordingly, the central interrupt control unit **220** of FIG. **21** is configured to optimally deal with system management interrupts that may be received from a system management source such as a power management unit **990**. The central controller **302** of central interrupt control unit **220** manages system management interrupts by designating one of the processing units **202-1** through **202-*m*** as the master SMM (system management mode) handler. The master SMM processing unit may be designated via system configuration. All SMM interrupt requests (i.e., the SMI signal from power management unit **990**) are routed to the master SMM processing unit. The remaining processing units **202-1** through **202-*m*** are referred to as slave processing units. FIG. **22** is a flow diagram that illustrates the operation of the master SMM processing unit as well as the slave SMM processing units.

Having one processing unit in system management mode and the other processing units executing normal code could cause system problems, such as an active processing unit accessing a peripheral just turned off by the SMM master processing unit. When the master SMM processing unit determines that its actions will have an effect on the other processors in the system, the master processing unit issues a command that causes the central controller **302** to assert an SMI to all other CPUs in the system. A field of the control register **650**D (or a separate register of CPU registers unit **650**) for each CPU channel **606** is provided that, per processing unit, holds a SMM code/flag value for each slave SMM processing unit. Initially, this flag indicates "HOLD" to each slave SMM processing unit. Upon entry of SMM and seeing the HOLD code in that register, each slave SMM processing unit will pause and poll its corresponding SMM code/flag field until it changes. When the master SMM processing unit processes the requested actions for system management (powering down a peripheral, for example), it changes the status of the other processing units SMM code flags with a special command (i.e., note that normally one processing unit cannot affect other processing units channel registers), which will allow them to continue SMM code execution. It is noted that the central controller **302** is configured to allow the master processing unit to write the SMM code flags of the other processing units CPU channel registers during the system management mode. Possible flag values could be "RETURN", "CONFIG CHANGE", "SHUTDOWN", etc. The slave SMM processing units can then determine the best possible actions to take in order to respond correctly to the new code flag value. Note that the master SMM processing unit and slave SMM processing units can be sharing the same SMM code space. Therefore the SMM code must identify master/slave status, via software control.

Referring collectively to FIGS. **20** and **21**, when power management unit **990** asserts the system management interrupt, the central controller **302** (FIG. **3**) of central interrupt control unit **220** asserts the SMI input of processing unit **202-1**, which is the designated SMM master for this system. This causes the processing unit **202-1** to begin execution of the SMM service routine during step **332**. During step **333**, the processing unit **202-1** determines the action required (in accordance with the specific SMM code provided by the system programmer), and determines whether the required actions may have an effect upon the other processing units in the system during step **334**. If the requested action will not affect any other processing units, the SMI service routine is completed during step **335**, and the processing unit **202-1** subsequently returns to normal operation.

If, on the other hand, the requested action during the system management service routine may have an effect with respect to the operation of the other processing units, the processing unit **202-1** causes the code/flag register within each of the other CPU channels to indicate that the other processing units should hold (step **336**). It is noted that a code/flag register (or field) for each slave processing unit is embodied within the CPU channel registers unit **650** of each CPU channel **606**. Subsequently, the processing unit **202-1** causes the central controller **302** to provide an SMI to each of the slave processing units **202-2** through **202-*m*** during step **337**. The processing unit **202-1** subsequently completes its requested system management mode operations (step **339**), and causes the code/flag registers of the slave processing units **202-2** through **202-*m*** to indicate that a change has occurred in the system configuration (step **339**). The system management interrupt routine for the processing unit **202-1** then completes, and the processing unit **202-1** returns to normal operation.

When the central controller **302** asserts the SMI signals to the slave processing units **202-2** through **202-*m***, each of the processing units **202-2** through **202-*m*** begins execution of the SMI service routine during step **340**. During execution of the SMI service routine (step **341**), each of the processing units **202-2** through **202-*m*** identify themselves as slaves, and consequently read their respective code/flag registers using the CPU ID registers **902-2** through **902-*m***, respectively, in a manner as described previously for accessing the configuration registers **650** of each CPU channel. If the code/flag register indicates that the respective processing unit should hold, the processing unit effectively remains idle until the code register indicates some other status. For example, the code register of the processing units **202-2** through **202-*m*** may indicate that the respective processing units should return to their normal execution with no further action required (step **343**). Similarly, the code/flag register may instruct the respective processing unit to shut down (step **344**). It is noted that the master processing unit **202-1** or the central controller **302** may have previously set the code/flag register of the particular processing unit to indicate "return" or "shutdown". If the code/flag register indicates that a configuration parameter within the system has

changed (step 345), the I/O permission map for the respective processing units is reconfigured to reflect the system change. It is noted that the reconfiguration of the I/O permission map for each processing unit is accomplished in a conventional manner.

In accordance with the symmetrical multiprocessing system as described above, efficient management of system interrupts is achieved while maintaining broad compatibility. Interrupt management is attained by way of a centralized interrupt control unit. The interrupt control unit advantageously allows for the expansion of each interrupt pin by setting the interrupt control unit in a cascade mode. Furthermore, the central control unit is responsive to specialized interrupt cycles which allows I/O devices and/or bus bridge devices to cause initiation of an interrupt without requiring a dedicated interrupt line. The central interrupt control unit further allows each interrupt to be prioritized independently of its associated vector ID, and prevents the occurrence of spurious interrupts by providing a programmable latency timer which causes the central interrupt control unit to delay its response to End Of Interrupt (EOI) instructions. An auto-chaining technique is further implemented by the central interrupt control unit to sequentially provide broadcast interrupts to various processing units based on their current task priority values. Finally, the central interrupt control unit further handles system management interrupts (SMIs) from sources such as power management units and ensures proper system operation even if the requested system management function affects operations being carried by other processing units.

Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

What is claimed is:

1. A multiprocessing computer system comprising:

a plurality of processing units;

a plurality of peripheral devices, wherein each of said peripheral devices is configured to generate a respective interrupt signal;

a plurality of interrupt channels coupled to said plurality of peripheral devices, wherein each interrupt channel is configured to detect an assertion of an interrupt of a corresponding peripheral device, and wherein each interrupt channel further includes a programmable storage unit for storing a vector and a priority value associated with said interrupt of said corresponding peripheral device;

a plurality of CPU channels coupled to said plurality of processing units, wherein each CPU channel is configured to receive interrupt information to control assertions of a corresponding processor interrupt, wherein said interrupt information includes said vector and priority value corresponding to each of several selected interrupts of said peripheral devices that have been asserted;

a central controller coupled to said plurality of interrupt channels and to said plurality of CPU channels, wherein said central controller is configured to route said interrupt information corresponding to said each of several interrupts that have been asserted from any of said plurality of interrupt channels to a designated CPU channel, and wherein said central controller is configured to prioritize several selected co-pending interrupts provided to said designated CPU channel in accordance with said priority values, whereby said designated processing unit is interrupted by a higher priority interrupt before being interrupted by a lower priority co-pending interrupt;

a status register coupled to said central controller configured to store information indicative of whether a particular interrupt signal is idle, wherein said central controller is configured to control the status of said status register according to interrupt signals dispatched to at least one of said plurality of processing units and an End Of Interrupt command executed by said at least one of said plurality of processing units; and

a programmable latency timer coupled to said central controller, wherein said central controller is configured to use a value within said programmable latency timer to control a time at which said status register is updated in response to an End Of Interrupt command.

2. An interrupt controller comprising:

a plurality of interrupt input terminals configured to receive a plurality of interrupt signals;

a central controller coupled to said input terminals configured to prioritize said plurality of interrupt signals and configured to provide said interrupt signals to at least one processing unit;

an I/O interrupt controller coupled between said plurality of interrupt input terminals and said central controller, said I/O interrupt controller being configured to monitor a status of said plurality of interrupt signals, to selectively respond to an assertion of any of said plurality of interrupt signals and to provide an indication of said assertion to said central controller;

a storage unit coupled to said central controller configured to store information indicative of whether a particular interrupt signal is idle, wherein said central controller is configured to control a content of said storage unit according to interrupt signals dispatched to said processing unit and an End Of Interrupt command executed by said processing unit; and

a programmable latency timer coupled to said central controller, wherein said central controller is configured to use a value within said programmable latency timer to control a time at which said storage unit is updated in response to an End Of Interrupt command, and

wherein said latency timer is programmable through an I/O bus.

3. An interrupt controller as recited in claim 2 wherein said central controller is configured to reset a particular interrupt to an idle state a predetermined time after said End Of Interrupt command.

4. An interrupt controller as recited in claim 2 wherein said I/O interrupt controller is configured to monitor the status of an interrupt if said storage unit indicates that said interrupt is in an idle state.

5. An interrupt controller as recited in claim 2 further including a processor interrupt generator coupled to said central controller, said processor interrupt generator including a plurality of interrupt output terminals coupled to a plurality of processing units and configured to dispatch interrupt signals to said plurality of processing units.

6. An interrupt controller as recited in claim 2 further including a decoder coupled between an I/O bus and said programmable latency timer, said decoder being configured to detect an End Of Interrupt command and of enabling said programmable latency timer in response to the detection of said End Of Interrupt command.

7. An interrupt controller as recited in claim 2, wherein said storage unit is a status register.

* * * * *

US005956516A

United States Patent [19]

Pawlowski

[11] Patent Number: 5,956,516

[45] Date of Patent: Sep. 21, 1999

[54] **MECHANISMS FOR CONVERTING INTERRUPT REQUEST SIGNALS ON ADDRESS AND DATA LINES TO INTERRUPT MESSAGE SIGNALS**

[75] Inventor: **Stephen S. Pawlowski**, Beaverton, Oreg.

[73] Assignee: **Intel Corporation**, Santa Clara, Calif.

[21] Appl. No.: **08/997,103**

[22] Filed: **Dec. 23, 1997**

[51] **Int. Cl.**[6] **G06F 9/46**

[52] **U.S. Cl.** **395/733**; 395/868; 395/309

[58] **Field of Search** 395/733, 736, 395/739, 868, 309

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,626,985	12/1986	Briggs	395/800.4
4,734,882	3/1988	Romagosa	395/733
5,701,496	12/1997	Nizar et al.	395/739
5,727,217	3/1998	Young	395/733
5,764,997	6/1998	Gulick	395/733
5,828,891	10/1998	Benayoun et al.	395/742

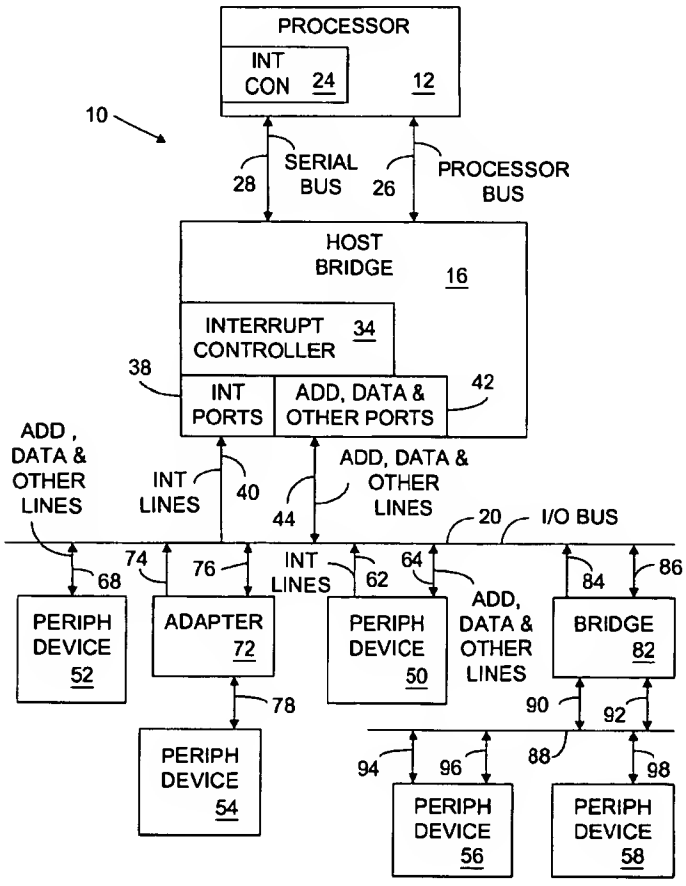
Primary Examiner—Gopal C. Ray

Attorney, Agent, or Firm—Alan K. Aldous

[57] **ABSTRACT**

In one embodiment of the invention, an apparatus includes address and data ports to receive an interrupt request signal in the form of address signals and data signals. The apparatus also includes decode logic to receive at least some of the address signals and data signals and provide a decoded signal at one of several decode output lines of the decode logic. A redirection table includes a send pending bit that is set responsive to the decode signal. In another embodiment, an apparatus includes dedicated interrupt ports to receive an interrupt request signal. The apparatus also includes address and data ports capable of receiving an interrupt request signal in the form of address signals and data signals, and decode logic to provide a decode signal at one of several decode output lines in response to reception of the interrupt request signal in the form of address signals and data signals. A redirection table includes a send pending bit to be set in response to either the interrupt request signal at the dedicated interrupt ports or in response to the decode signal.

16 Claims, 6 Drawing Sheets



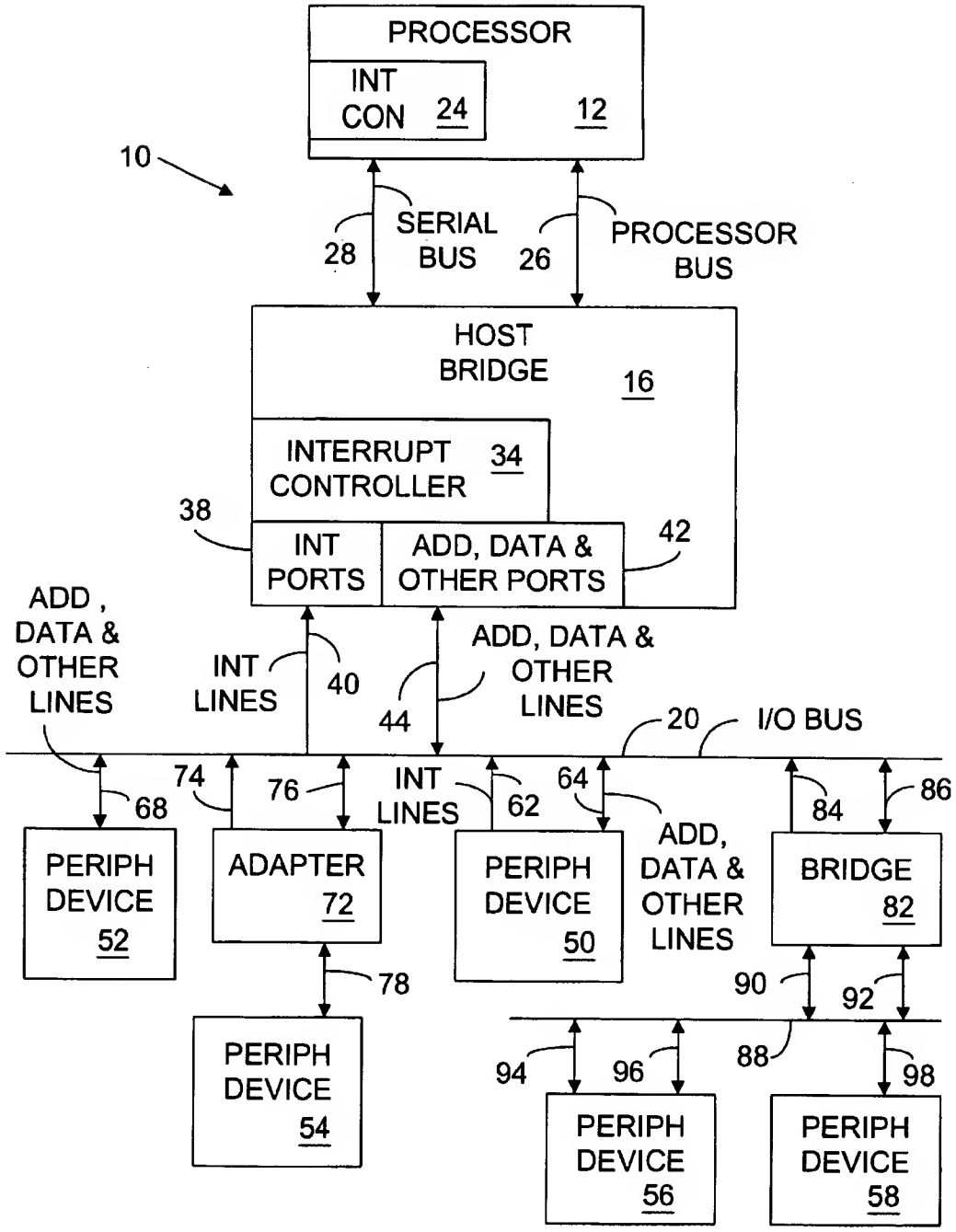

PROCESSOR
INT CON 24
12
10
28
SERIAL BUS
26
PROCESSOR BUS
HOST BRIDGE 16
INTERRUPT CONTROLLER 34
38
INT PORTS
ADD, DATA & OTHER PORTS
42
ADD , DATA & OTHER LINES
INT LINES
40
44
ADD, DATA & OTHER LINES
20
I/O BUS
68
74
76
INT LINES
62
64
84
86
PERIPH DEVICE 52
ADAPTER 72
PERIPH DEVICE 50
ADD, DATA & OTHER LINES
BRIDGE 82
78
90
92
PERIPH DEVICE 54
94
96
88
98
PERIPH DEVICE 56
PERIPH DEVICE 58

FIG. 1

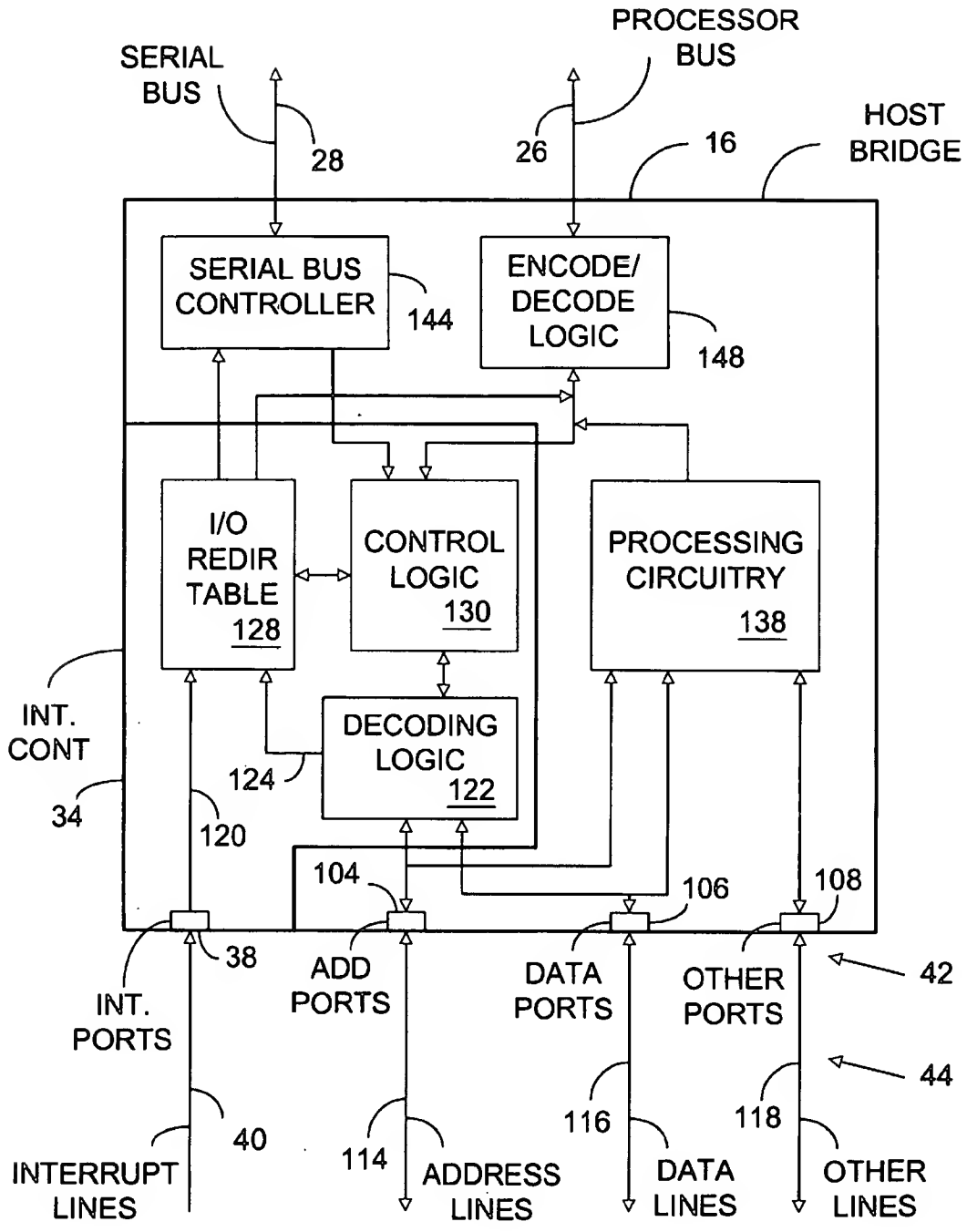

SERIAL BUS
28
PROCESSOR BUS
26
16
HOST BRIDGE
SERIAL BUS CONTROLLER
144
ENCODE/ DECODE LOGIC
148
I/O REDIR TABLE 128
CONTROL LOGIC 130
PROCESSING CIRCUITRY 138
DECODING LOGIC 122
INT. CONT
34
124
120
104
106
108
38
INT. PORTS
ADD PORTS
DATA PORTS
OTHER PORTS
42
44
40
116
118
INTERRUPT LINES
114
ADDRESS LINES
DATA LINES
OTHER LINES

FIG. 2

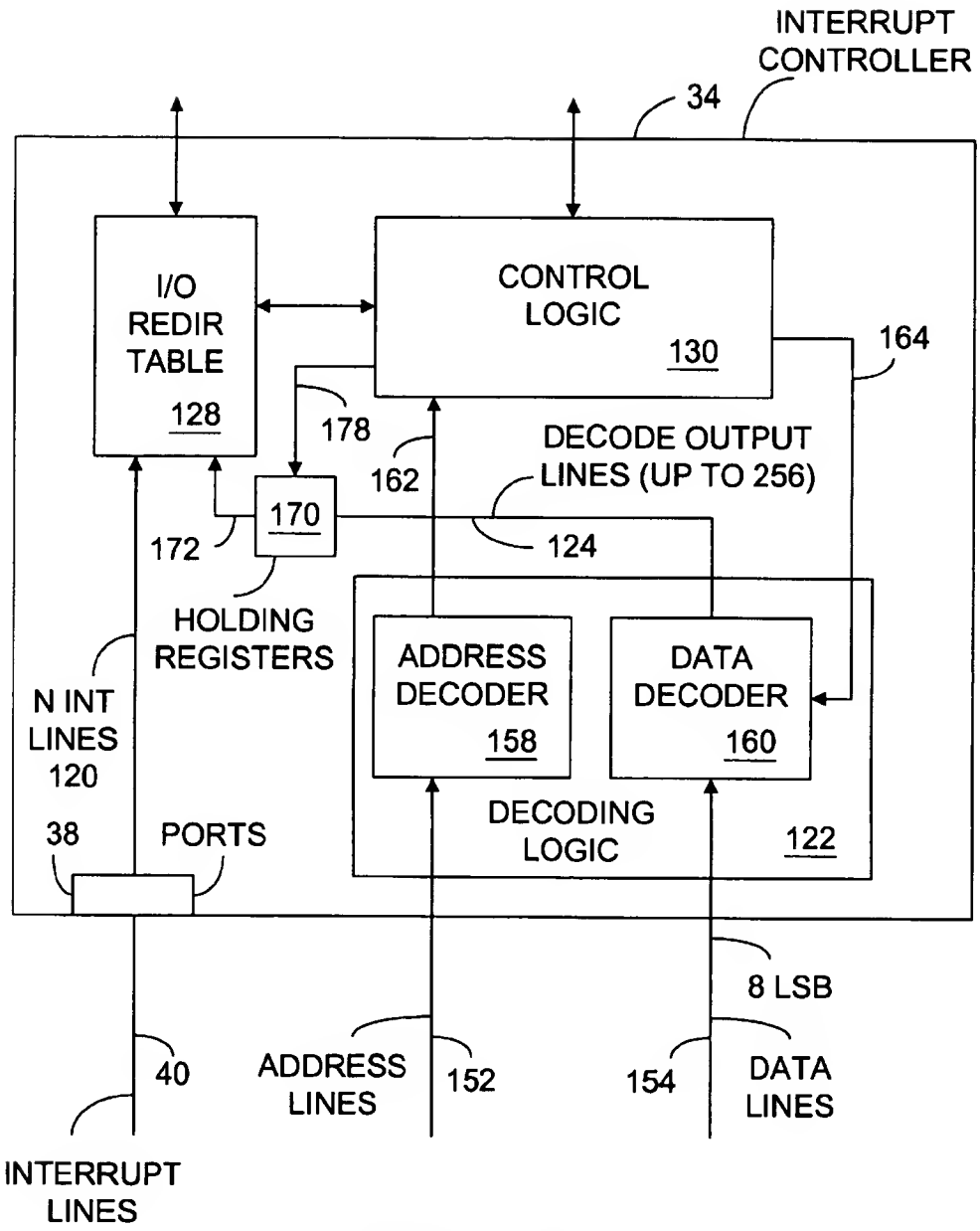

INTERRUPT CONTROLLER
34
I/O REDIR TABLE
128
CONTROL LOGIC
130
164
178
162
DECODE OUTPUT LINES (UP TO 256)
170
172
124
HOLDING REGISTERS
N INT LINES 120
ADDRESS DECODER
158
DATA DECODER
160
DECODING LOGIC
122
38
PORTS
8 LSB
40
ADDRESS LINES
152
154
DATA LINES
INTERRUPT LINES

FIG. 3

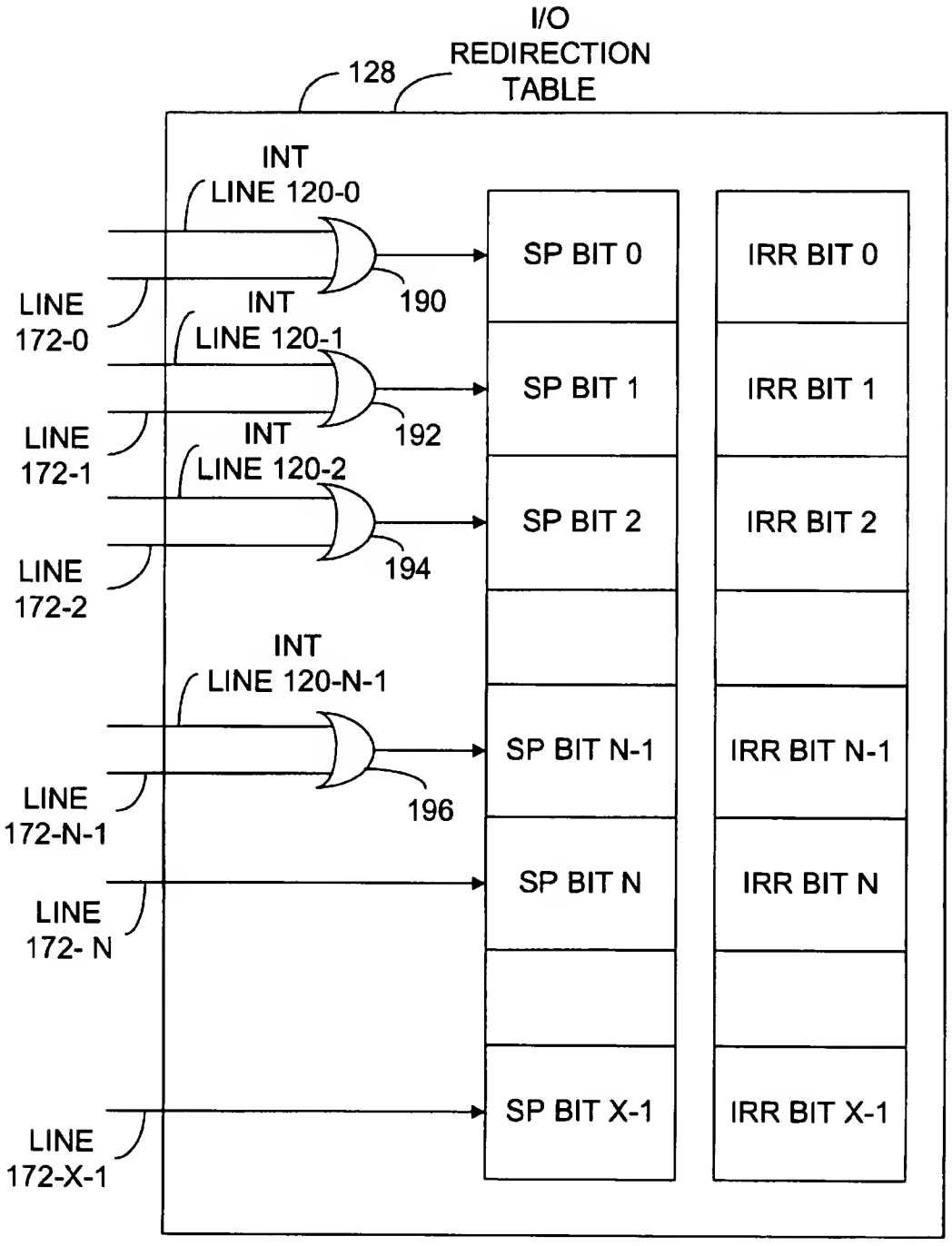

I/O
REDIRECTION
TABLE
128
INT
LINE 120-0
INT
LINE 120-1
INT
LINE 120-2
INT
LINE 120-N-1
190
192
194
196
LINE
172-0
LINE
172-1
LINE
172-2
LINE
172-N-1
LINE
172- N
LINE
172-X-1
SP BIT 0
SP BIT 1
SP BIT 2
SP BIT N-1
SP BIT N
SP BIT X-1
IRR BIT 0
IRR BIT 1
IRR BIT 2
IRR BIT N-1
IRR BIT N
IRR BIT X-1

FIG. 4

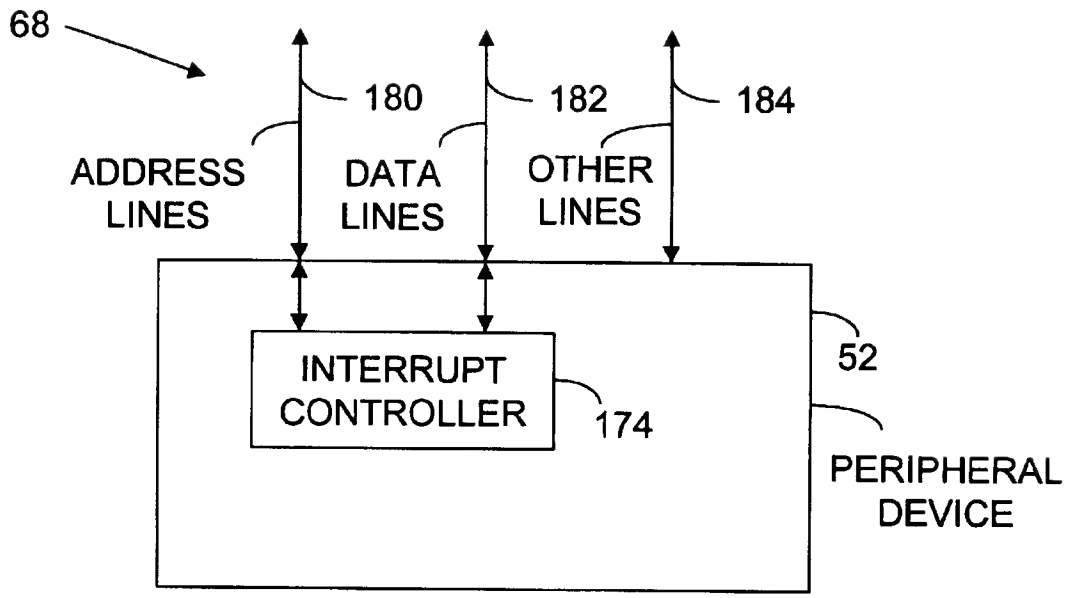

68
180
182
184
ADDRESS LINES
DATA LINES
OTHER LINES
INTERRUPT CONTROLLER
174
52
PERIPHERAL DEVICE

FIG. 5

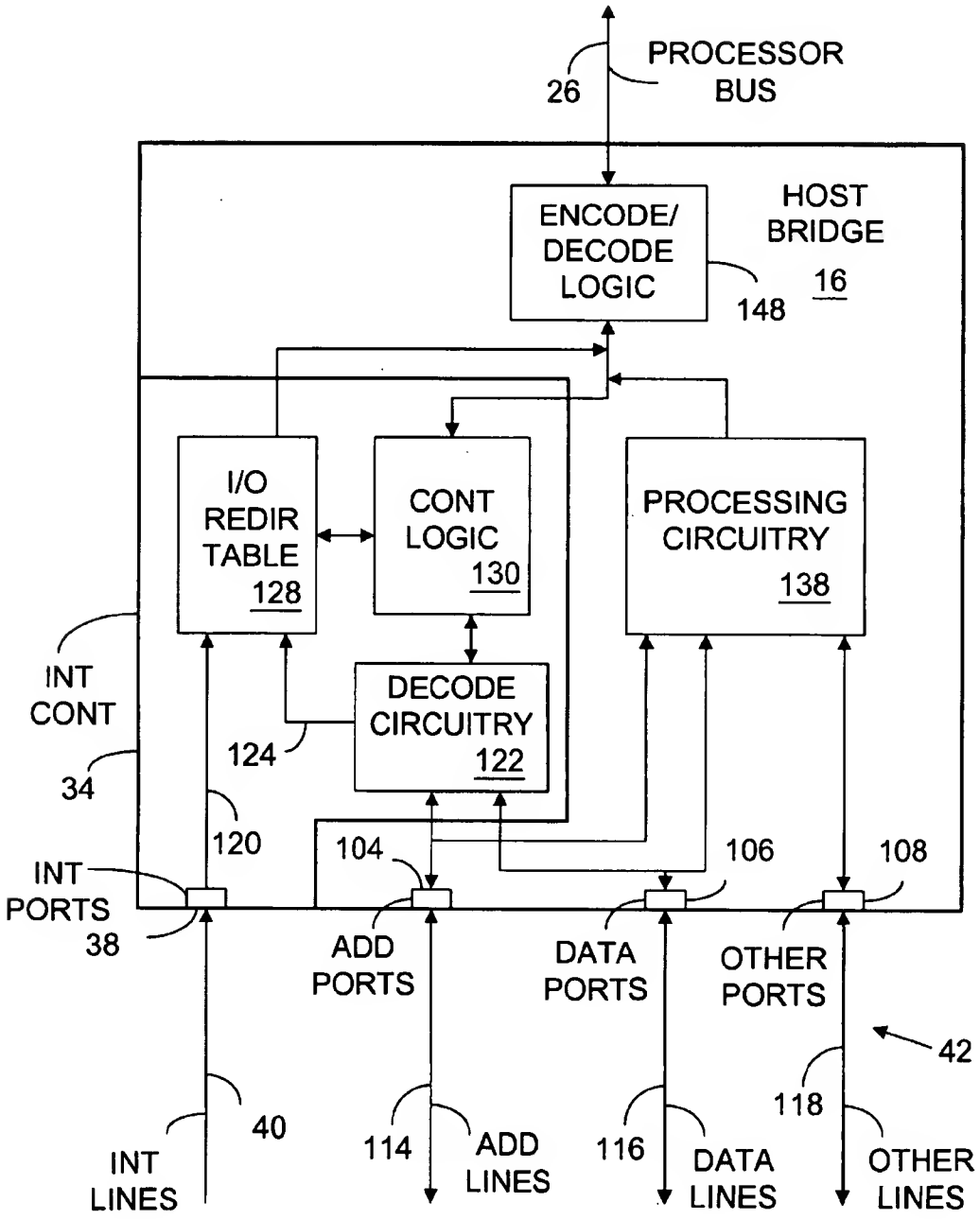

26
PROCESSOR BUS
ENCODE/ DECODE LOGIC
148
HOST BRIDGE
16
I/O REDIR TABLE
128
CONT LOGIC
130
PROCESSING CIRCUITRY
138
DECODE CIRCUITRY
122
124
INT CONT
34
120
INT PORTS
38
104
106
108
ADD PORTS
DATA PORTS
OTHER PORTS
42
118
40
INT LINES
114
ADD LINES
116
DATA LINES
OTHER LINES

FIG. 6

MECHANISMS FOR CONVERTING INTERRUPT REQUEST SIGNALS ON ADDRESS AND DATA LINES TO INTERRUPT MESSAGE SIGNALS

BACKGROUND OF THE INVENTION

1. Technical Field of the Invention

The present invention relates to interrupts in a computer system.

2. Background Art

A peripheral component interconnect (PCI) Local Bus Specification (Revision 2.1) ("PCI bus specification") has been developed to define a PCI bus. The PCI bus specification defines an interconnect mechanism and transfer protocol for devices on the bus. Additions or changes to the PCI specification are occasionally made. However, a guiding principle of the PCI specification is that of backward compatibility, wherein newer PCI systems will support older PCI peripheral devices.

Various devices including input and/or output (I/O) peripheral devices may seek to interrupt a processor in a computer system. When associated with a PCI bus, the devices are sometimes referred to as PCI agents. To interrupt a processor, the PCI agent may send one or more of interrupt request signals INTA#, INTB#, INTC#, or INTD# to an interrupt controller. The interrupt controller responds by providing an interrupt message to a processor. The interrupt controller receives the interrupt request signal through interrupt input pins. The interrupt input pins are sometimes called interrupt request (IRQ) pins, which are connected through IRQ lines to the PCI bus. There may be an interrupt router between the peripherals and the interrupt controller.

There are two types of signaling semantics for interrupt signals received by interrupt controllers: (1) edge triggered interrupt semantics and (2) level triggered interrupt semantics. With edge triggered interrupts, every time an edge (e.g., positive going edge) is detected at an interrupt input pin, the interrupt controller triggers an interrupt event. A problem with edge triggered interrupts is that the interrupt controller may miss an edge of a second interrupt if it occurs before a first interrupt is serviced. Accordingly, in the case of edge triggered interrupts, typically only one peripheral device is connected to the interrupt input pin.

With level triggered interrupts, a particular logical voltage level (e.g., a logical high voltage) at the interrupt input pin causes the interrupt controller to trigger an interrupt event. In the case of level triggered interrupts, more than one peripheral device may provide interrupt request signals to an input pin. However, the voltage level at the interrupt input pin provided by multiple peripheral devices is not different than the voltage level that is provided by only one peripheral device. Accordingly, the interrupt controller cannot determine how many peripheral devices are providing an interrupt request signal merely by sensing the voltage level at the interrupt input pin. In response to detecting a change to the particular voltage level at the interrupt input pin, an interrupt message is sent to a processor and a state bit is set in an I/O redirection table in the interrupt controller. The state bit is reset when an end-of-interrupt (EOI) signal is received by the interrupt controller. If an interrupt signal having the particular voltage level is still detected at the interrupt input port after the EOI is received, another interrupt message is sent to a processor.

Interrupt controllers have a limited number of interrupt input pins. Under the present technology, as more peripheral devices are added to a computer system, the number of interrupt input pins will need to be increased or peripheral devices may need to wait longer for service of interrupts.

Accordingly, there is a need for an improved system for providing interrupt requests from peripheral devices to processors.

SUMMARY OF THE INVENTION

In one embodiment of the invention, an apparatus includes address and data ports to receive an interrupt request signal in the form of address signals and data signals. The apparatus also includes decode logic to receive at least some of the address signals and data signals and provide a decoded signal at one of several decode output lines of the decode logic. A redirection table includes a send pending bit that is set responsive to the decode signal.

BRIEF DESCRIPTION OF THE DRAWINGS

The invention will be understood more fully from the detailed description given below and from the accompanying drawings of embodiments of the invention which, however, should not be taken to limit the invention to the specific embodiments described, but are for explanation and understanding only.

FIG. **1** is a block diagram representation of a computer system including a host bridge according to one embodiment of the present invention.

FIG. **2** is a block diagram representation of one embodiment of the host bridge in the system of FIG. **1**.

FIG. **3** is a block diagram representation of one embodiment of the interrupt controller in the system of FIG. **1**.

FIG. **4** is a block diagram representation of one embodiment of send pending bits and related circuitry in the I/O redirection table of FIGS. **2** and **3**.

FIG. **5** is a block diagram representation of an exemplary peripheral device.

FIG. **6** is a block diagram representation of an alternative embodiment of a host bridge in the system of FIG. **1**.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

Referring to FIG. **1**, a computer system **10** includes a processor **12**, a host bridge **16**, and an I/O bus **20**, which may be implemented according to PCI specifications. Processor **12** includes an interrupt controller **24**, which may be an advance programmable interrupt controller (APIC). Processor **12** is coupled to host bridge **16** through a processor bus **26** and a serial bus **28**, which may be an APIC serial bus. In some embodiments, host bridge **16** is referred to as a North Bridge and processor bus **26** is referred to as a front side bus or parallel bus.

Serial bus **28** may provide interrupt messages from an interrupt controller **34** in host bridge **16** to interrupt controller **24**. Interrupt controller **34** may be an APIC. Serial bus **28**, which may include two data conductors and a clock signal conductor, may also provide signals from processor **12** to interrupt controller **34**, such as end-of-interrupt (EOI) signals. In multi-processor systems, serial bus **28** may also be used in lowest priority interrupt destination arbitration according to known techniques.

Host bridge **16** includes dedicated interrupt (e.g., IRQ) ports **38** through which interrupt request signals (e.g., IRQ signals) are received from interrupt request lines **40**. Interrupt ports **38** may be pins, other structure, or simply con-

ductors. Interrupt controller **34** receives the interrupt request signals from ports **38**. In one embodiment, interrupt ports **38** are considered part of interrupt controller **34** (and, therefore, also part of host bridge **16**). In another embodiment, interrupt ports **38** are considered part of host bridge **16**, but not interrupt controller **34**. The difference is not important so long as interrupt controller **38** receives interrupt request signals.

Host bridge **16** also includes address, data & other ports **42** through which address, data, and other signals are received from or provided to address, data & other lines **44**. Ports **42** may be pins, other structure, or any other conductor. Ports **38** and **42** may be simply continuations of lines **40** and **44**. Interrupt controller **34** receives at least some of the address, data, and/or other signals passing through ports **42**. Some or all of the address, data, and other signals received at ports **42** are used in host bridge **16** and elsewhere for various purposes other than interrupts. Lines **40** and **44**, as well as the various other lines described herein, may be conventional conductor traces or various other forms of conductors. Depending on the embodiment of the invention, lines **40** and **44** maybe considered part of or separate from bus **20**.

System **10** includes peripheral devices that may interrupt processor **12** through providing interrupt request signals to interrupt controller **34**. Examples of peripheral devices and related interconnections are illustrated in FIG. **1**. Peripheral device **50** is coupled to I/O bus **20** through interrupt line(s) **62** and through address, data & other lines **64**. To interrupt processor **12**, peripheral device **50** provides an interrupt request signal (e.g., INTA#) on interrupt line(s) **62**. The interrupt request signal is passed through bus **20** and interrupt lines **40** to interrupt ports **38**. Interrupt controller **34** responds to the interrupt request signal by providing an appropriate interrupt message to processor **12** or another processor (not shown in FIG. **1**).

Peripheral device **52** is coupled to I/O bus **20** through address, data & other lines **68**, but not through interrupt line(s). To interrupt processor **12**, peripheral device **52** provides an interrupt request signal on address, data & other lines **68**. In one embodiment of the invention, discussed in greater detail below, the interrupt request signal involves a PCI write cycle. The interrupt request signal is passed through bus **20** and address, data & other lines **44**. Interrupt controller **34** responds to the interrupt request signal by providing an appropriate interrupt message to processor **12** or another processor (in the case of a multi-processor system not shown in FIG. **1**).

Accordingly, host bridge **16** may provide interrupt messages to processor **12** or another processor in response to interrupt request signals from two types of peripheral devices. A first type of peripheral device (e.g., peripheral device **50**) provides interrupt request signals (e.g., INTA#) through dedicated interrupt line(s). The interrupt request signals are received by interrupt controller **34** through interrupt ports **38**. A second type of peripheral device (e.g., peripheral device **52**) provides interrupt request signals (e.g., including a PCI write cycle) through, for example, address and data lines. The interrupt request signals are received by interrupt controller **34** through address, data & other lines **44**.

Peripheral devices **54**, **56**, and **58** illustrate other possible interfaces between peripheral devices and bus **20**. Peripheral device **54** is coupled to bus **20** through an adapter **72**. Adapter **72** may conduct interrupt signals through line(s) **74** and address, data & other signals through lines **76**. Interrupt request signals that are provided on conductors **74** are passed by bus **20** to interrupt lines **40**. Peripheral device **54** is like peripheral device **52** in that it provides interrupt request signals through address, data and other signals, not through an interrupt line(s) **74**. Therefore, in the case of peripheral device **54**, there are no interrupt request signals on interrupt lines **74**. However, a peripheral device like peripheral device **50** could be connected to adapter **72**. In that case, adapter **74** would include interrupt signals on line(s) **74**. Alternatively, some adapters could include only lines **76** and not line(s) **74**. Peripheral devices **56** and **58** are coupled to bus **20** through a bridge **82**. Interrupt request signals are conducted through lines **94**, **90**, and **84**. Address, data & other signals are conducted through lines **98**, **96**, **92**, and **86**.

FIG. **2** shows details of one embodiment of host bridge **16**. Address, data & other ports **42** includes address ports **104**, data ports **106**, and other ports **108**. Address, data, and other lines **44** include address lines **114**, data lines **116**, and other lines **118**, which conduct address signals, data signals, and other signals (e.g., enable signals), respectively.

An interrupt request signal on interrupt lines **40** is provided through ports **38** and conductors **120** to I/O redirection table **128** or other processing circuitry. In response thereto, interrupt controller **34**, including I/O redirection table **128**, provides an interrupt message to a processor. The interrupt message may be provided through serial bus **28** through serial bus controller **144** or through processor bus **26** through encode/decode logic **148**. In the case of sending the interrupt message over processor bus **26**, processor **12** would include decode circuitry to detect the interrupt message and interrupt controller **24** would understand the message.

In response to receiving an interrupt request signal, at least a portion of which is in the form of address signals, interrupt controller **34** provides an interrupt message to serial bus **28** or processor bus **26**. In one embodiment, host bridge **16** can direct the interrupt message either through serial bus **28** or processor bus **26** depending on a bit in control logic **130**.

The interrupt message over processor bus **26** could include an address identifying the processor to receive the interrupt. Host bridge **16** could include lowest priority redirection circuitry to redirect the interrupt to the processor having the lowest priority in the case of a multi-processor system. The circuitry could keep track of task priorities of the various processors in a multi-processor system. Interrupt controller **34** or other circuitry in host bridge **16** could detect whether processor **12** includes serial bus capabilities and/or the ability to accept interrupt messages by processor bus **26**. In the case where processor **12** does not include an interrupt controller and decode circuitry that understands an interrupt message over processor bus **26**, interrupt controller **34** could direct the interrupt message over serial bus **28** rather than over processor bus **26**. Host bridge **16** may include queues (not shown) to hold various interrupt signals and other signals. Interrupt controller **34** may include queues to hold interrupt request signals. Control logic **130** assists in various functions of interrupt controller **34**.

An interrupt request signal may be provided in the form of address and data signals (and perhaps other signals) through ports **42** and captured by interrupt controller **34**. In such a case, decoding logic **122** decodes all or part of the address and data signal bits as being an interrupt request signal. In one embodiment, decoding logic **122** provides a decoded signal on conductors **124**. In one embodiment, the

decode signal may be an assertion or a deassertion signal. The interrupt request assertion/deassertion signals on conductors **124** may be the same as the interrupt request signals on conductors **120**. In that case, I/O redirection table **128** could treat the signals identically.

Referring to FIG. **3**, in one embodiment, decoding logic **122** includes an address decoder **158** and a data decoder **160**. If a particular address or an address within a particular range is received, address decoder **158** provides a signal to control logic **130** on conductor(s) **162** indicating that an interrupt request signal is being provided to interrupt controller **34** through address and data lines **152** and **154**, which are connected to lines **114** and **116**. In one embodiment, an address indicating an interrupt request signal includes a base plus an offset. As an example, the base could be FEC00000h (where h=hex). The offset could be 20h. The base may be programmable by the processor, operating system, or other hardware or software. Control logic **130** provides an enabling signal on conductor(s) **164** to data decoder **160**. In one embodiment, data decoder **160** decodes the 8 least significant bits (LSBs) of the data signal and asserts one of X decode output lines **124**, depending on the state of the data bits. If there are 8 data bits, there may be up to 256 decode output lines **124**.

Holding registers **170** include a register for each one of decode output lines **124**. Each of the holding register holds the voltage state on a corresponding one of decode output lines **124**. In turn, lines **172** provide signals representing the voltage state held in holding registers **170**. A holding register is set (e.g., has a logic high voltage) through an assertion signal on the corresponding one of lines **124** and is reset through a deassertion signal on the corresponding one of lines **124**. The difference between the assertion and deassertion signals may be merely opposite polarity. In one embodiment, a different address on conductors **152** controls whether an assertion or deassertion signal is provided on decode output lines **124**. In another embodiment, different data signals on conductors **154** control whether an assertion or deassertion signal is provided on a particular one of decode output lines **124**.

Referring to FIGS. **3** and **4**, lines **172** include lines **172-0**, **172-1**, . . . , **172-X-1**, each connected to a different one of holding registers **170**. Interrupt lines **120** include interrupt lines **120-0**, **120-1**, **120-2**, . . . , **120-N-1**, each connected to a different one of interrupt ports **38**. In the embodiment of FIG. **4**, I/O redirection table **128** includes X entries, which each include a "send pending" (SP) bit (which may be called a delivery status bit). When an SP bit is set, an interrupt message is sent to a processor. SP bits **0**-N-1 are set (e.g., to a logic high voltage) when the output of a corresponding OR-gate **190**, **192**, **194**, . . . , **196** is asserted. The OR-gates have inputs of one of lines **120** and one of lines **172**. Accordingly, an interrupt signal to either one of ports **38** or to decoding logic **122** may cause one of SP bits **0**-N-1 to be set. For example, SP bit **0** is set when either interrupt line **120-0** or line **172-0** is set. (The OR-gates could be replaced with other logic if SP bits are set through a low voltage. There could be inverters between interrupt ports **38** and the OR gates). SP bits N-X-1 are set when a corresponding one of lines **172**-N–**172**-X-1 is asserted. In this way, there may be a greater number of SP bits than interrupt ports **38**. (Note that in the some embodiments and in certain circumstances, the states of the SP bits **0**-X-1 may be controlled by signals other than those from lines **120** or **172**).

Interrupt controller **34** may support scalability for edge triggered interrupt request signals. In the case of edge triggered interrupts on lines **152** and **154**, data decoder **160** assserts one of lines **124**. The corresponding one of holding registers **170** is set, causing a correponding one of lines **172** to be asserted. Assertion of one of lines **172** causes the corresponding one of SP bits is set. When the SP bit is set, the particular one of holding registers **170** is reset through conductors **178**. This I/O redirection entry may be then entered into the interrupt delivery rotation scheme to be delivered at the appropriate time. There is no need to initiate an interrupt request deassertion register operation when the interrupt event is removed, because the activation of the signal itself may indicate that one and only one interrupt event will be signaled. As with the input pin scheme, the SP bit of an interrupt defined as edge triggered may be reset when the interrupt has been successfully delivered on the associated message mechanism. If multiple interrupt request assertion register operations are received to the same I/O redirection table entry before the interrupt has been delivered to the destination only one interrupt event may be detected. This behavior is consistent with the dedicated pin scheme.

With respect to level triggered interrupts, when a device signals an interrupt for a line that is shared by multiple devices, that device may issue an interrupt request operation on the first activation of the interrupt. When the interrupt signal goes inactive, the device may issue an interrupt request deassertion message to interrupt controller **34**. Interrupt controller **34** maintains the activation of the corresponding holding register bit until the deassertion message is received. The constraint of this mechanism is that both the device collecting the input events and the interrupt controller are cognizant that the interrupt request is configured as a level triggered interrupt event. For these events, the interrupt request deassertion register transactions may be required for correct operation. Signals on lines **116** or **118** may indicated whether an edge or level triggered interrupt signal is involved.

In the embodiment of FIG. **4**, **1/0** redirection table **128** also includes interrupt request register (IRR) bits **0**, **1**, . . . , X-1, which are used in the case of level triggered interrupts. The SP bit is reset when the IRR hit is set. The IRR bit is set when an interrupt message is accepted hy the processor. Either a level assert message is issued and not retried on processor bus **16** or a message on serial bus **28** is accepted. The IRR bit is reset when an EOI message is received. For both serial and parallel bus delivery, the IRR bit is reset with a write to the corresponding EOI register, the vector of which matches the vector field of the redirection entry.

When an interrupt is serviced, a deassertion signal is provided by the peripheral device to decode logic **122**. If after the IRR bit is reset, the corresponding holding register is set, then there is another interrupt waiting to be acknowledged. The corresponding SP bit is then set.

FIG. **5** illustrates details of one embodiment of peripheral device **52**. Address, data & other lines **68** include address lines **180**, data lines **182**, and other lines **184**. An interrupt controller **174** provides interrupt request signals to at least some of the bits of address lines **180**. The interrupt request signal may also include bits on data lines **182** and/or other lines **184**. In one embodiment, interrupt controller **174** includes a data register(s) the contents of which control whether peripheral device **52** sends interrupt request signals in the form of an interrupt signal to a dedicated interrupt port or in the form of address and data signals, and particular details regarding the signals.

An advantage of the invention is that level triggered interrupts on interrupt lines **40** may be replaced by write

cycle messages or other address signal based messages. In one embodiment, the write cycle message may identify the origin of the interrupt request. Further, the number of send pending bits may be easily increased without the adding dedicated interrupt lines.

Interrupt controller **34** may support multiple interrupt request signal input mechanisms. However, in order to avoid any race conditions that may occur, in one embodiment, only one mechanism per interrupt request signal is supported at a given time. The interaction of the various arrival times and rates may be identical to the dedicated port (e.g., pin) approach. Multiple activations of an event from a device will elicit the interrupt request assertion/deassertion signal which may provide a model consistent with the operation of the dedicated port.

Each interrupt controller may have a unique address for configurability and any access to this address space, regardless of the initiating resource may reach the final destination. As an example, if a system contains two I/O buses, the first contains the interrupting device and the second contains the interrupting controller. The interrupting device, through the unique address of the interrupting controller, may be capable of directing an interrupt request assertion signal to the interrupting controller. Note that this messaging scheme does not require a 'sidecar' path for interrupts that is different than the path to main memory. Signaling the interrupt request assertion signal may have the effect of flushing any previous write transactions.

Additional Information and Embodiments

The specification does not describe or illustrate various well known components, features, and conductors, a discussion of which is not necessary to understand the invention and inclusion of which would tend to obscure the invention. Furthermore, in constricting an embodiment of the invention, there are design tradeoffs and choices, which would vary depending on the embodiment. There are a variety of ways of implementing the illustrated and unillustrated components.

The borders of the boxes in the figures are for illustrative purposes and do not restrict the boundaries of the components, which may overlap. The relative size of the illustrative components does not to suggest actual relative sizes. Arrows show principle data flow in one embodiment, but not every signal, such as requests for data flow. As used herein "logic" does not mean that software control cannot be involved. The term "conductor" is intended to be interpreted broadly and includes devices that conduct although they also have some insulating properties. There may be intermediate components or conductors between the illustrated components and conductors.

The interrupt message provided by interrupt controller **34** to interrupt controller **24** may be somewhat altered in host bridge **16**, processor bus **26**, and/or serial bus **28** prior to it being received by interrupt controller **24**. For example, bits of the interrupt message provided by interrupt controller **34** could be inverted or encoded. Address bits could be added by encode/decode logic or other circuitry.

In one embodiment, host bridge **16** does not include the capability to send interrupt messages over processor bus **26**. In that embodiment, conductors might not connect I/O redirection table **128** to encode/decode logic **148**. As shown in FIG. **5**, in another embodiment, host bridge **16** does not include the capability to send interrupt messages over serial bus **28**. In that embodiment, serial bus controller **144** and associated conductors are not included in host bridge **16**.

In one embodiment, a signal on processor bus **26** is two phase signal. In the first phase, if an Aa3# bit is 0, the interrupt transaction type is fixed (directed); if the Aa3# bit is 1, the type is redirected or EOI. In the second phase, Ab5# and Ab6# bits of 00 indicate physical destination mode, and Ab5# and Ab6# bits of 01 indicate logical destination mode. Ab5# and Ab6# bits of 11 indicate an EOI. Aa3# and Ab6# bits of 0 and 1 and Aa3#, Ab5#, and Ab6# bits of 110 are reserved.

The holding registers and SP bits may be in parallel with respect to conductors **124**.

Interrupt controller **34** does not have to be part of host bridge **16**. There may be an interrupt router between the peripheral devices (interrupting agents or PCI devices) and the interrupt controller. Decode logic **122** may be outside interrupt controller **34**.

The phrase "in one embodiment" means that the particular feature, structure, or characteristic following the phrase is included in at least one embodiment of the invention, and may be included in more than one embodiment of the invention. Also, the appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same one embodiment.

The term "connected" and "coupled" and related terms are used in an operational sense and are not necessarily limited to a direct connection or coupling. If the specification states a component or feature "may", "can", "could", or "might" be included or have a characteristic, that particular component or feature is not required to be included or have the characteristic. The term "responsive" includes completely or partially responsive.

Those skilled in the art having the benefit of this disclosure will appreciate that many other variations from the foregoing description and drawings may be made within the scope of the present invention. Accordingly, it is the following claims including any amendments thereto that define the scope of the invention.

What is claimed is:

1. An apparatus, comprising:

address and data ports to receive an interrupt request signal in the form of address signals and data signals;

decode logic to receive at least some of the address signals and data signals and to provide a decoded signal at one of several decode output lines of the decode logic; and

a redirection table including a send pending bit that is set responsive to the decode signal.

2. The apparatus of claim **1**, further including a holding register that are set in response to assertion of the decoding signal.

3. The apparatus of claim **1**, wherein the interrupt request signal is in the form of the address signals, the data signals, and other signals.

4. The apparatus of claim **1**, wherein the apparatus is a bridge.

5. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal;

address and data ports capable of receiving an interrupt request signal in the form of address signals and data signals;

decode logic to provide a decode signal at one of several decode output lines in response to reception of the interrupt request signal in the form of address signals and data signals; and

a redirection table including a send pending bit to be set in response to either the interrupt request signal at the dedicated interrupt ports or in response to the decode signal.

6. The apparatus of claim 5, wherein OR gates are positioned between the interrupt ports, some of the decode lines, and the redirection table.

7. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal from interrupt request lines;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

redirection and control circuitry coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

8. The apparatus of claim 7, wherein the interrupt request signal provided by the decode logic is identical to the interrupt request signal provided by the dedicated interrupt ports.

9. The apparatus of claim 7, wherein the redirecton and control circuitry includes an I/O redirection table.

10. The apparatus of claim 7, wherein the decode logic is included in an interrupt controller.

11. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal from interrupt request lines;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

an I/O redirection table coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

12. The apparatus of claim 11, wherein the interrupt request signal provided by the decode logic is identical to the interrupt request signal provided by the dedicated interrupt ports.

13. The apparatus of claim 11, wherein the decode logic is included in an interrupt controller.

14. A computer system, comprising:

a processor;

an I/O bus;

peripheral devices connected to the I/O bus; and

a bridge including:

dedicated interrupt ports to receive an interrupt request signal;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

redirection and control circuitry coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

15. The system of claim 14, wherein at least one of the peripheral devices provide interrupt request signals through the dedicated interrupt ports and at least one of the peripheral devices provides interrupt request signals in the form of write cycles through the address and data ports.

16. The system of claim 14, wherein at least one of the peripheral devices can provide address and data signals to the decode logic for decoding an interrupt request signal from them.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 5,956,516

DATED : September 21, 1999

INVENTOR(S) : Pawlowski

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In column 3, at line 13, delete "38" and insert --34--.

In column 3, at line 66, delete"74" and insert --72--.

In column 5, at line 29, delete "register" and insert --registers--.

In column 5, at line 62, delete "the"

In column 6, at line 2, delete "correponding" and insert--corresponding--.

In column 6, at line 4, delete "is" and insert--to be--.

In column 6, at line 34, delete "indicated" and insert--indicate--.

In column 6, at line 43, delete "16" and insert--26--.

In column 7, at line 4, delete "the".

In column 7, at line 34, delete "constricting" and insert--constructing--.

In column 7, at line 42, delete "to"

In column 7, at line 43, delete "principle" and insert--principal--.

In column 7, at line 66, insert "a"

In column 8, at line 47, delete "are" and insert--is--.

In column 8, at line 63, delete "and"

Signed and Sealed this

Twenty-fourth Day of April, 2001

Attest:

Nicholas P. Godici

Attesting Officer

NICHOLAS P. GODICI

Acting Director of the United States Patent and Trademark Office

US006249830B1

(12) **United States Patent**
Mayer et al.

(10) **Patent No.: US 6,249,830 B1**
(45) **Date of Patent: Jun. 19, 2001**

(54) **METHOD AND APPARATUS FOR DISTRIBUTING INTERRUPTS IN A SCALABLE SYMMETRIC MULTIPROCESSOR SYSTEM WITHOUT CHANGING THE BUS WIDTH OR BUS PROTOCOL**

(75) Inventors: **Dale J. Mayer**, Houston; **Sompong Paul Olarig**; **William F. Whiteman**, both of Cypress; **David F. Heinrich**, Tomball, all of TX (US)

(73) Assignee: **Compaq Computer Corp.**, Houston, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/532,109**

(22) Filed: **Mar. 21, 2000**

Related U.S. Application Data

(63) Continuation of application No. 09/243,235, filed on Feb. 2, 1999, now Pat. No. 6,041,377, which is a continuation of application No. 08/699,912, filed on Aug. 20, 1996, now abandoned.

(51) **Int. Cl.**[7] **G06F 13/00**

(52) **U.S. Cl.** **710/113**; 710/260

(58) **Field of Search** 710/113, 9, 260

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,125,093	6/1992	McFarland	395/725
5,283,904 *	2/1994	Carson et al.	710/266
5,379,434	1/1995	DiBrino	395/725
5,410,710	4/1995	Sarangdhar et al.	395/725
5,434,997	7/1995	Landry et al.	395/575
5,437,042	7/1995	Culley et al.	395/800
5,446,910	8/1995	Kennedy et al.	395/800
5,481,725	1/1996	Jayakumar et al.	395/725
5,530,891	6/1996	Gephardt	395/800
5,555,430	9/1996	Gephardt et al.	395/800
5,564,060	10/1996	Mahalingaiab et al.	395/871
5,568,649	10/1996	MacDonald et al.	395/868
5,613,126	3/1997	Schmidt	395/733
6,041,377 *	3/2000	Mayer et al.	710/113

FOREIGN PATENT DOCUMENTS

0 602 858 A1	6/1994	(EP)	13/26
0 685 798 A2	12/1995	(EP)	13/26

OTHER PUBLICATIONS

Hewlett Packard; "White Paper on Multiprocessing"; Jun. 1995; pp. 1–11.

Intel; "Pentium Processor Integrated APIC"; pp. 19–4 through 19–40; Apr. 1994.

Advanced Micro Devices and Cyrix Corporation; AMD web site; "The Open Programmable Interrupt controller (PIC) Register Interface Specification Revision 1.2"; Oct. 1995; pp. 1–24.

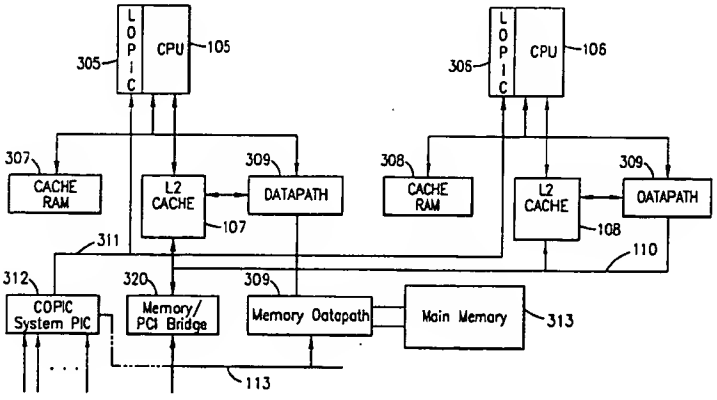
(List continued on next page.)

Primary Examiner—David A. Wiley
(74) *Attorney, Agent, or Firm*—Fletcher, Yoder & Van Someren

(57) **ABSTRACT**

A method for supporting multiple distributed interrupt controllers, desigoated as bus agents, in a symmetric multiprocessing system, which method includes the steps of assigning a unique identification number to each bus agent, receiving bus requests from the bus agents over four data lines in groups of four, and granting bus ownership to a selected one of the requesting bus agents. Similarly, a computer system that supports multiple distributed interrupt controllers, designated as bus agents, in a symmetric multiprocessing system, which computer system includes structure for assigning a unique identification number to each bus agent, four data lines for receiving bus requests from the bus agents in groups of four, and structure for granting bus ownership to a selected one of the requesting bus agents.

12 Claims, 9 Drawing Sheets



OTHER PUBLICATIONS

Cyrix Technical Brief; "OpenPIC™ Architecture"; pp. 1–2.
AMD News Release; "Advanced Micro Devices and Cyrix Jointly Develop OpenPIC™ Technology Standard"; Jun. 4, 1996; pp. 1–2.

Russo, A.P.: "The Alphaserver 2100 I/O Subsystem", Digital Technical Journal, vol. 6, No. 3, Jan. 1, 1994, pp. 20–28, XP000575573, p. 25, Right–Hand Column, Line 5–P. 26, Left–Hand Column, Line 46.

* cited by examiner

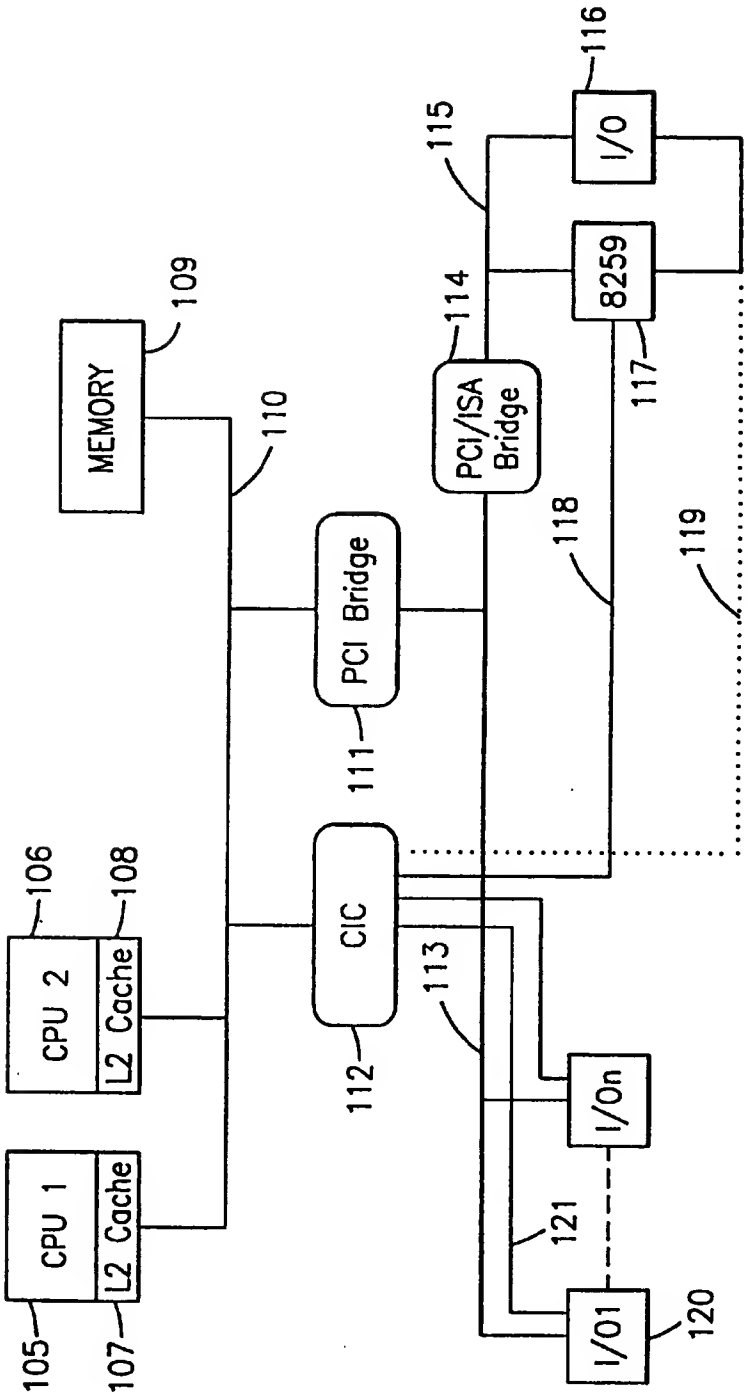


FIG. 1 (PRIOR ART)

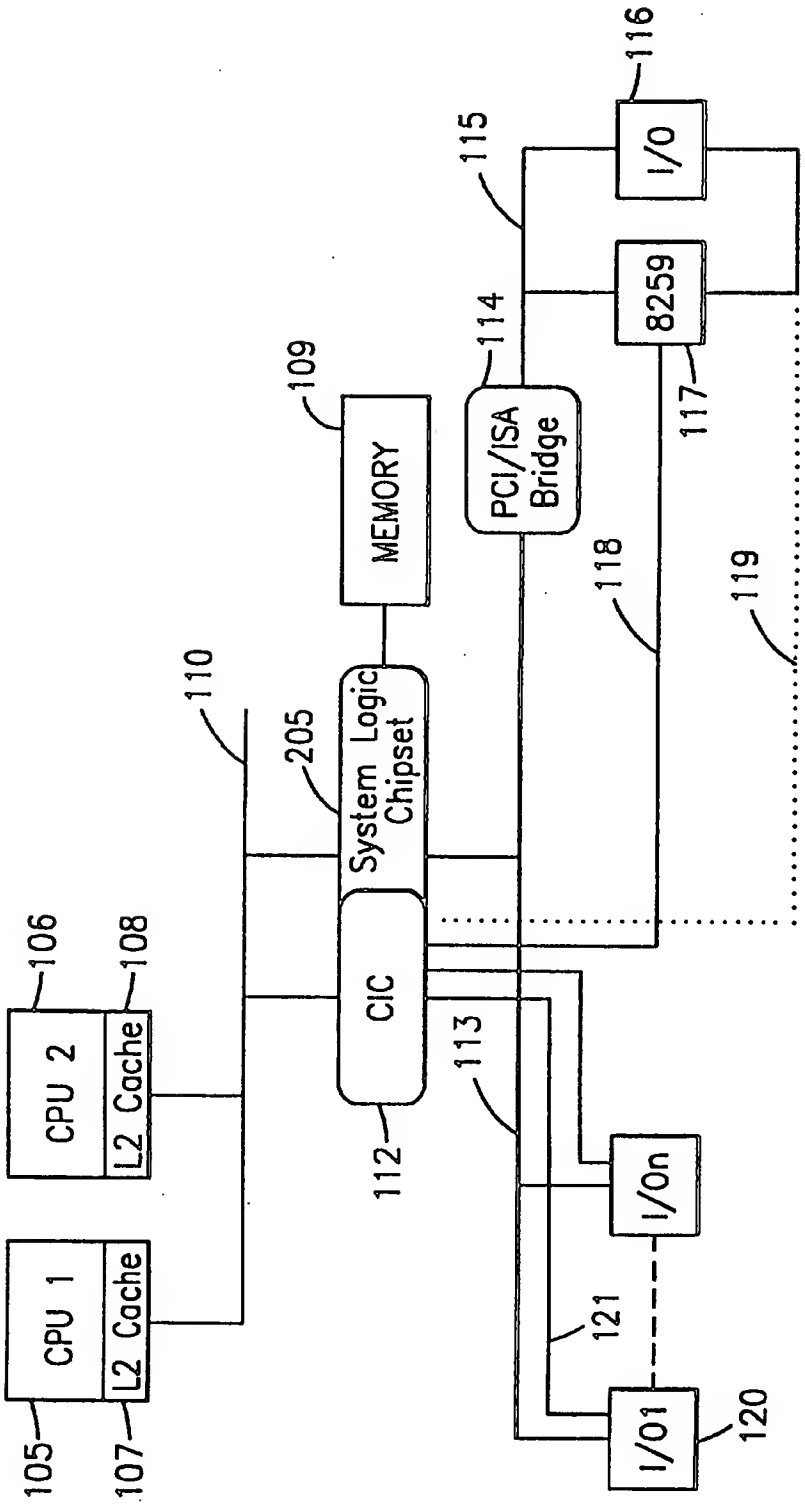


FIG. 2 (PRIOR ART)

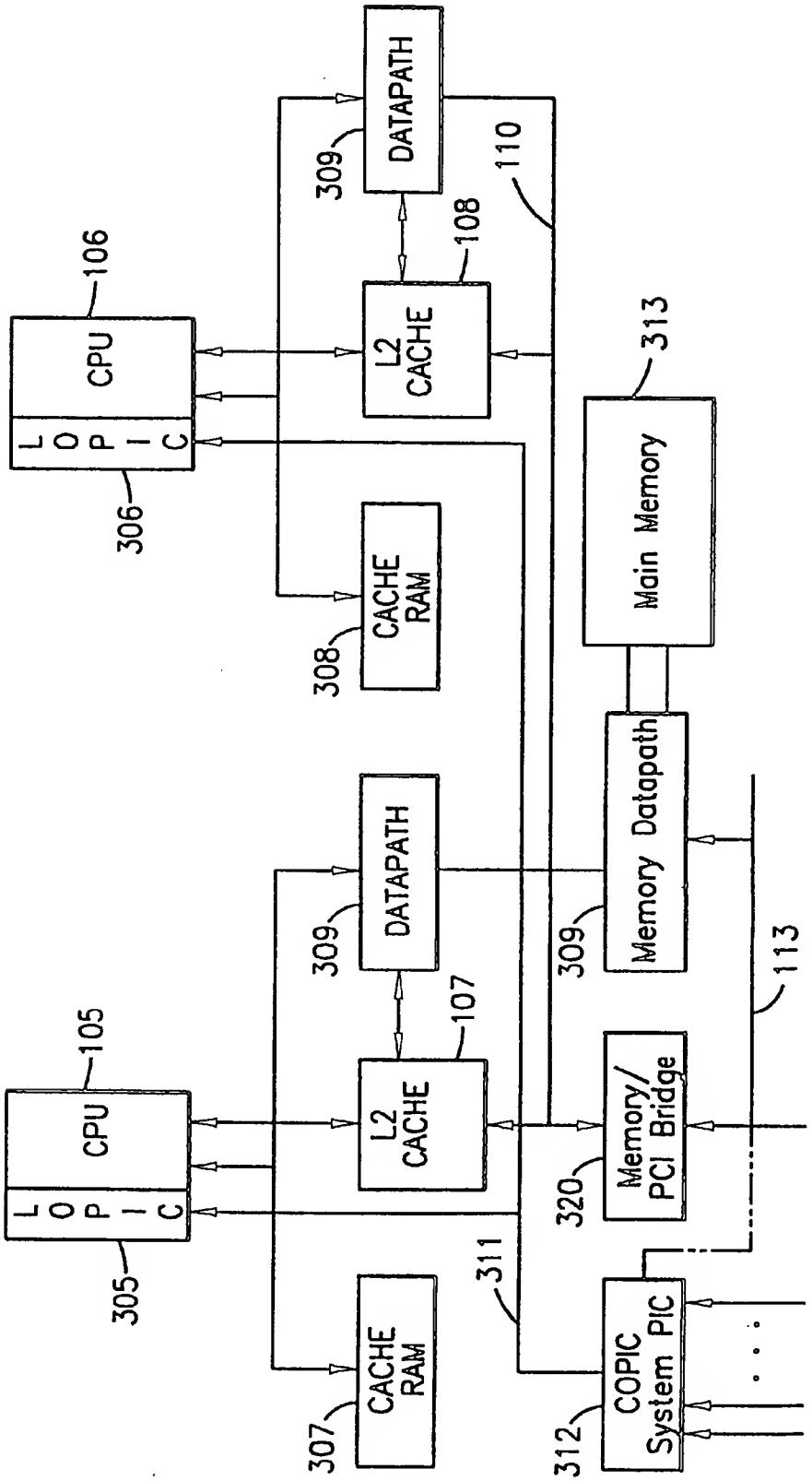

CPU
L O P I C
106
306
105
305
DATAPATH
309
L2 CACHE
108
107
CACHE RAM
308
307
110
Main Memory
313
Memory Datapath
Memory/ PCI Bridge
320
113
COPIC System PIC
312
311

FIG. 3

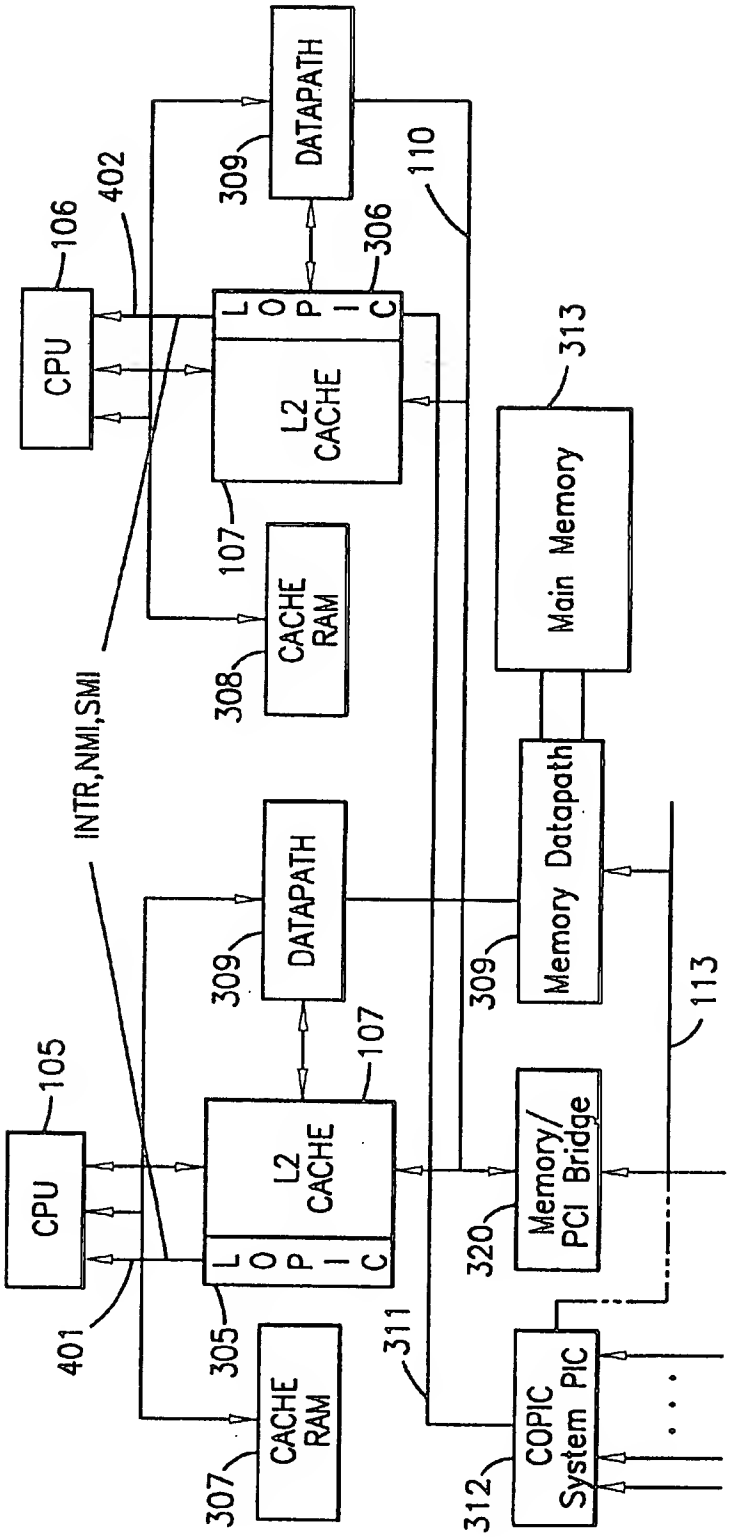

CPU
106
402
DATAPATH
309
110
306
LOPIC
L2 CACHE
107
CACHE RAM
308
313
Main Memory
INTR,NMI,SMI
DATAPATH
309
Memory Datapath
309
113
105
L2 CACHE
107
LOPIC
Memory/ PCI Bridge
320
401
305
311
CACHE RAM
307
COPIC System PIC
312

FIG. 4

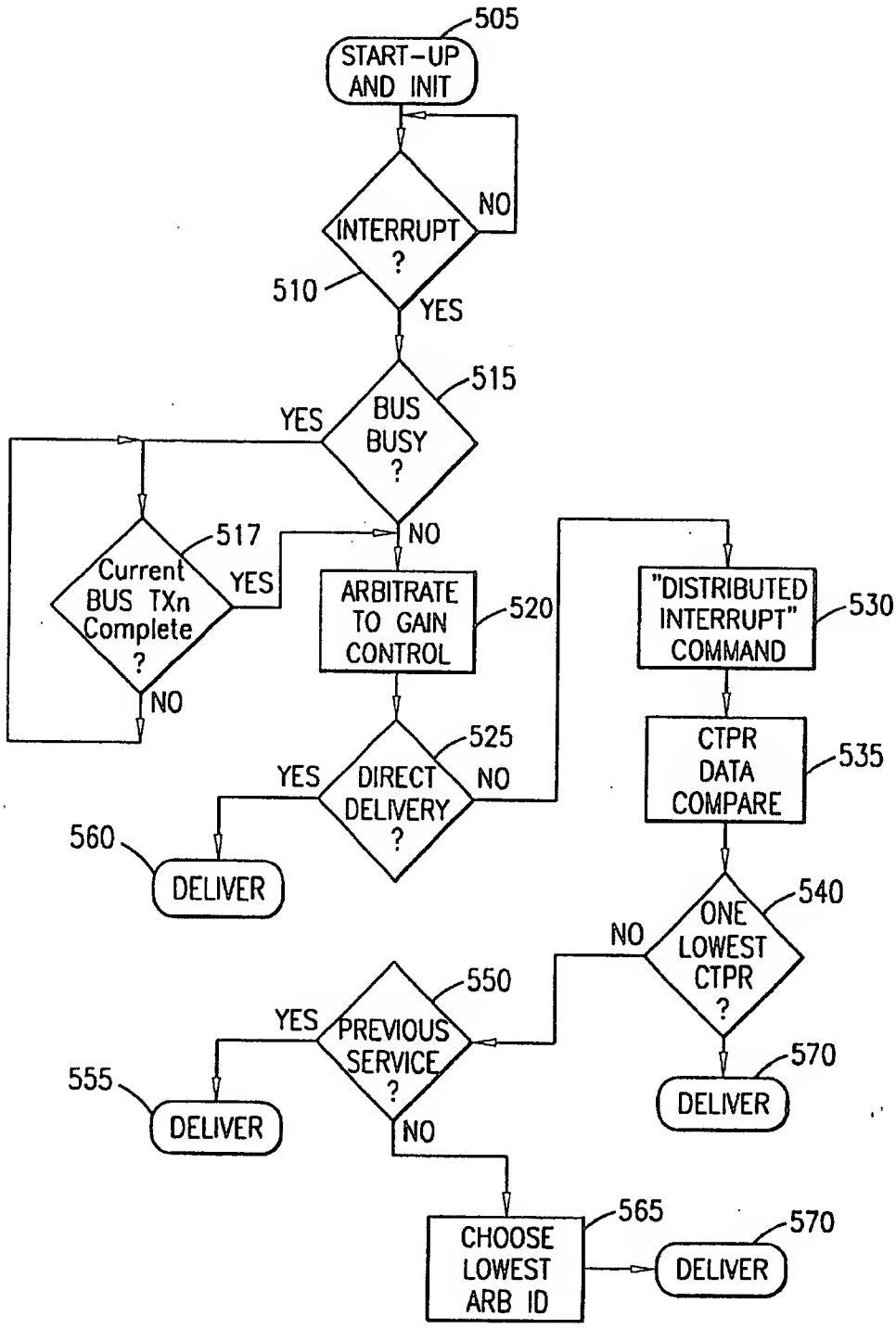

505
START-UP AND INIT
INTERRUPT ?
NO
510
YES
515
BUS BUSY ?
YES
NO
517
Current BUS TXn Complete ?
YES
NO
ARBITRATE TO GAIN CONTROL
520
"DISTRIBUTED INTERRUPT" COMMAND
530
CTPR DATA COMPARE
535
525
DIRECT DELIVERY ?
YES
NO
560
DELIVER
540
ONE LOWEST CTPR ?
NO
550
PREVIOUS SERVICE ?
YES
555
DELIVER
NO
570
DELIVER
565
CHOOSE LOWEST ARB ID
570
DELIVER

FIG. 5

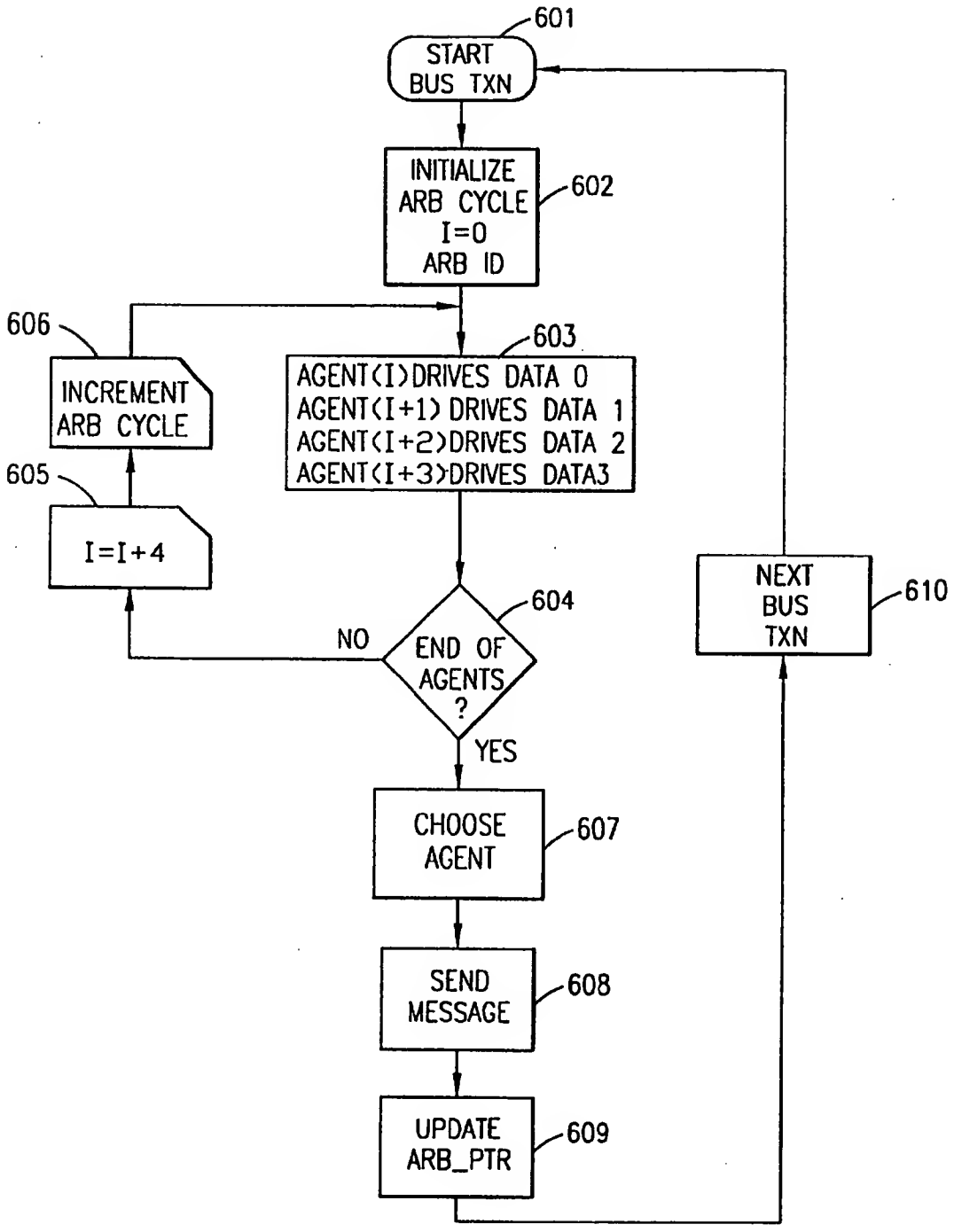

601
START BUS TXN
INITIALIZE ARB CYCLE I=0 ARB ID
602
606
603
INCREMENT ARB CYCLE
AGENT(I) DRIVES DATA 0
AGENT(I+1) DRIVES DATA 1
AGENT(I+2) DRIVES DATA 2
AGENT(I+3) DRIVES DATA3
605
I=I+4
604
NO
END OF AGENTS ?
YES
NEXT BUS TXN
610
CHOOSE AGENT
607
SEND MESSAGE
608
UPDATE ARB_PTR
609

FIG. 6

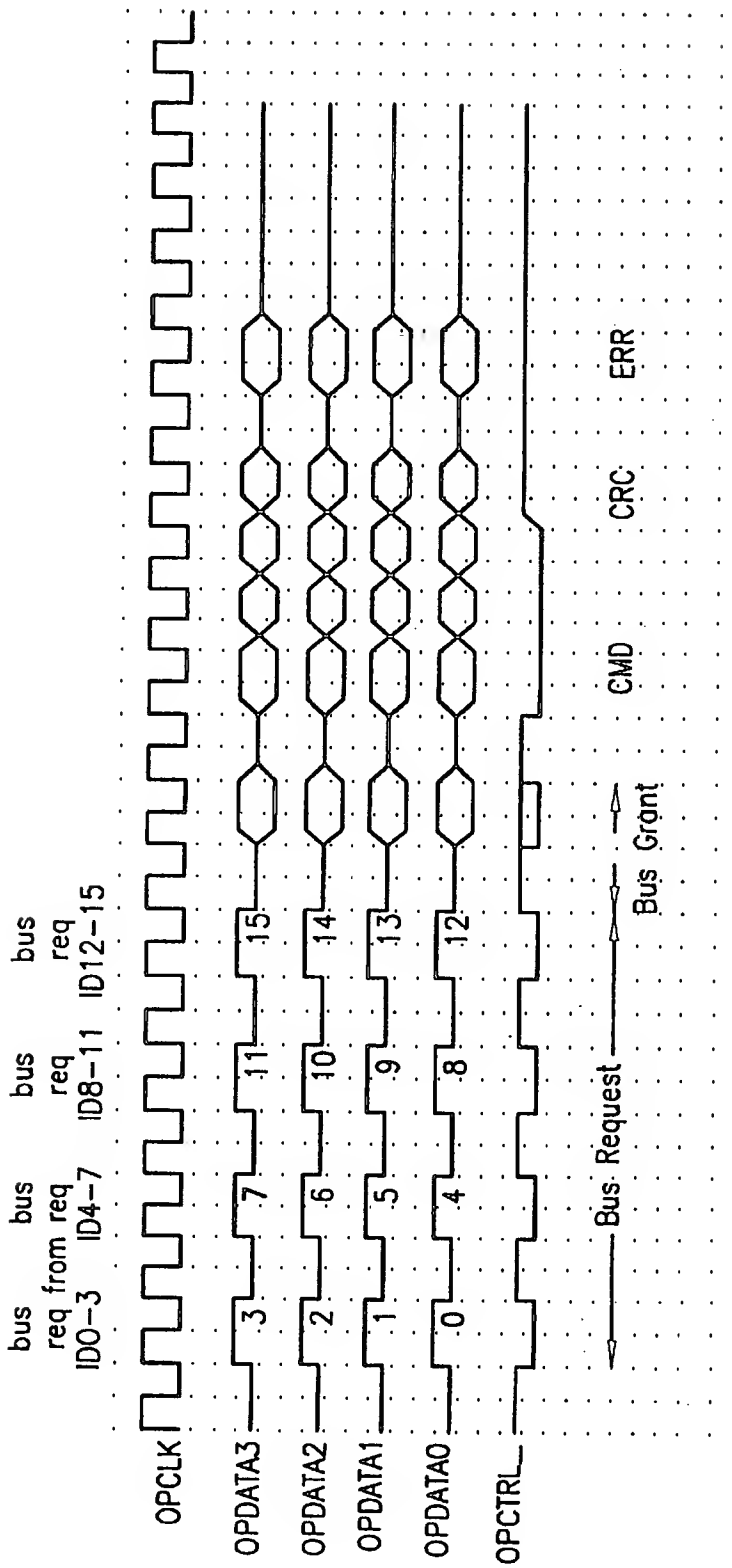

OPCLK
OPDATA3
OPDATA2
OPDATA1
OPDATA0
OPCTRL
bus req from req ID0–3
bus req ID4–7
bus req ID8–11
bus req ID12–15
3
7
11
15
2
6
10
14
1
5
9
13
0
4
8
12
Bus Request
Bus Grant
CMD
CRC
ERR
Arbitration Cycle if N>4

FIG. 7

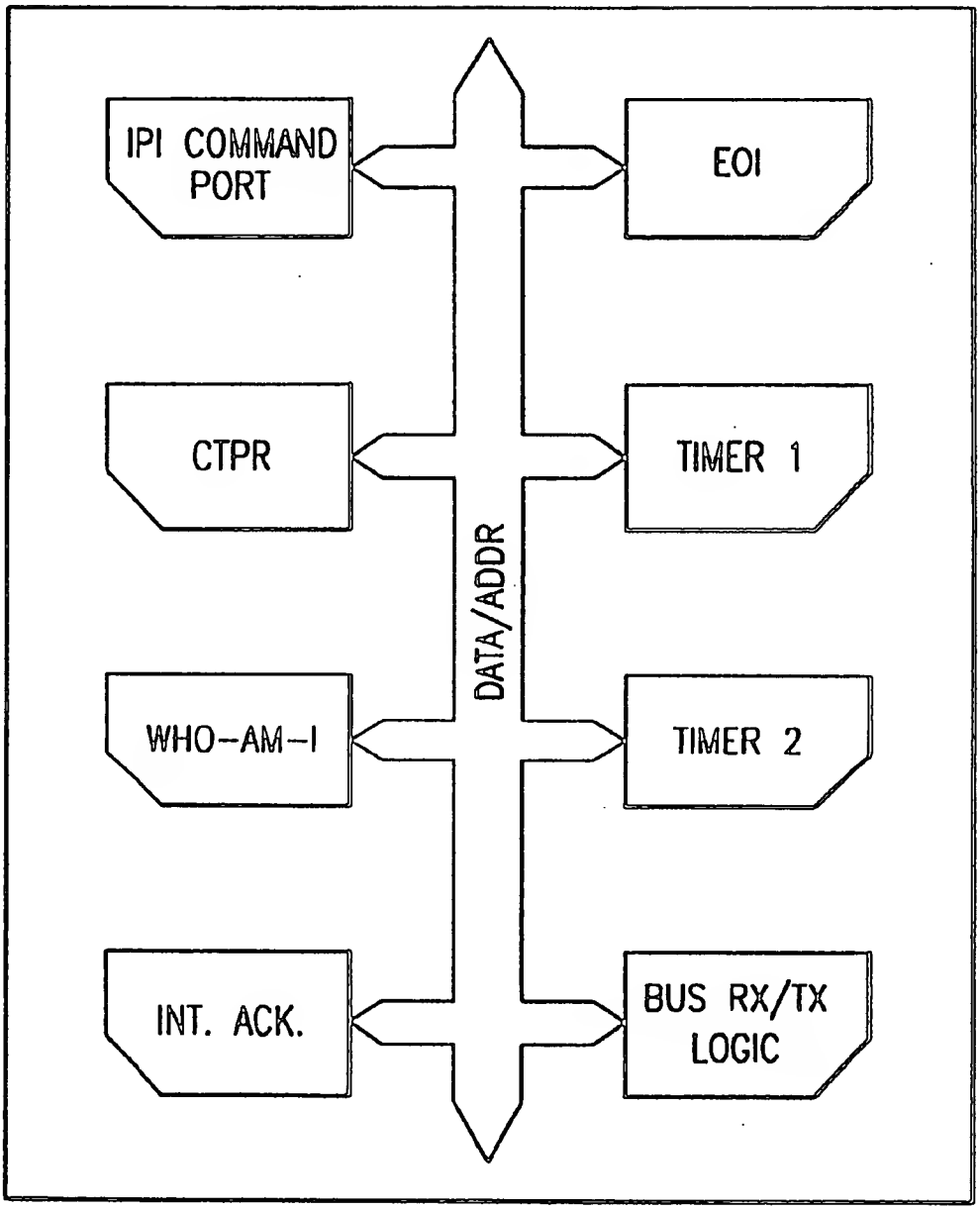

IPI COMMAND PORT
EOI
CTPR
TIMER 1
DATA/ADDR
WHO-AM-I
TIMER 2
INT. ACK.
BUS RX/TX LOGIC

FIG. 8

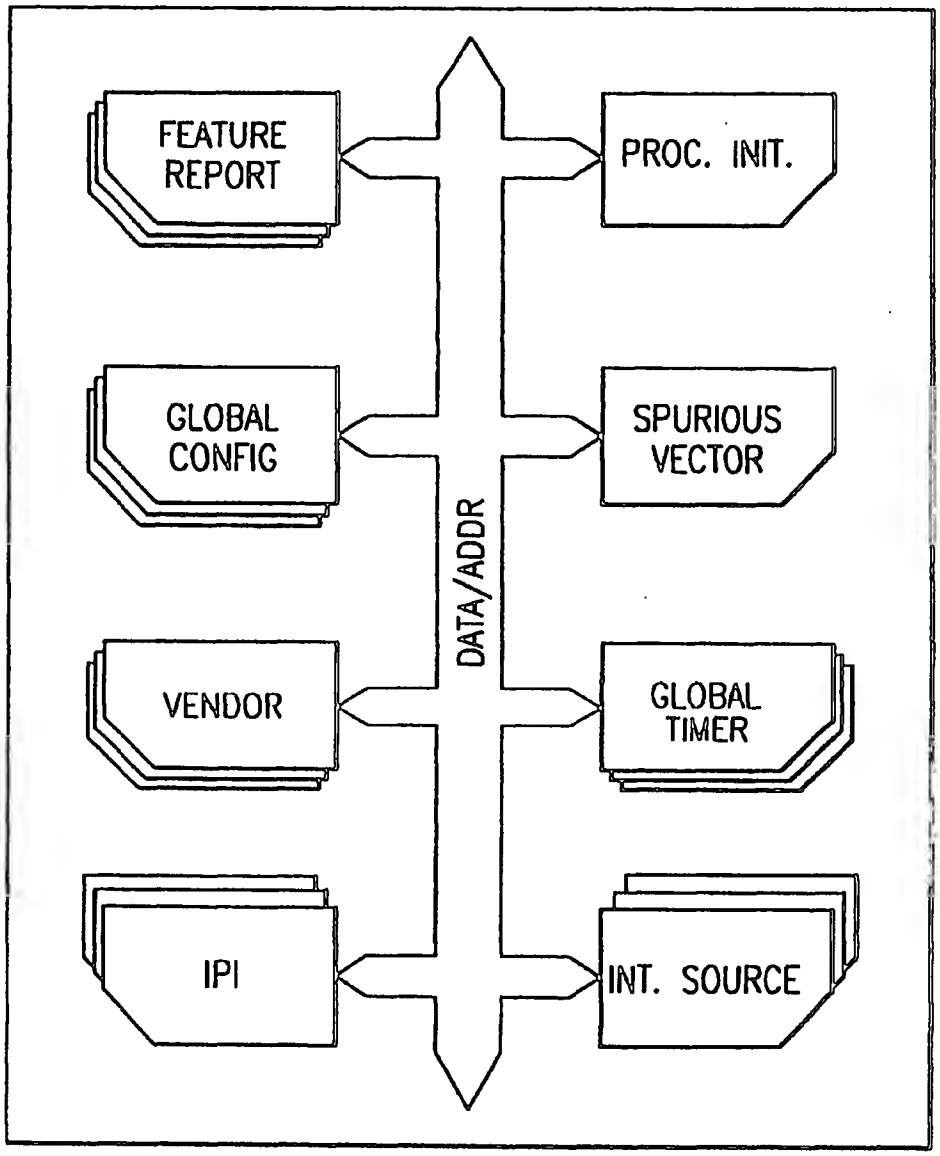

FEATURE REPORT
PROC. INIT.
GLOBAL CONFIG
SPURIOUS VECTOR
DATA/ADDR
VENDOR
GLOBAL TIMER
IPI
INT. SOURCE

FIG. 9

METHOD AND APPARATUS FOR DISTRIBUTING INTERRUPTS IN A SCALABLE SYMMETRIC MULTIPROCESSOR SYSTEM WITHOUT CHANGING THE BUS WIDTH OR BUS PROTOCOL

This application is a continuation of Ser. No. 09/243,235 filed on Feb. 2, 1999, now U.S. Pat. No. 6,041,377 which is a continuation of Ser. No. 08/699,912 filed on Aug. 20, 1996, now abandoned.

CROSS REFERENCE TO RELATED APPLICATION

This application hereby incorporates by reference the following co-assigned U.S. patent application, entitled "METHOD AND APPARATUS FOR DISTRIBUTING INTERRUPTS IN A SYMMETRIC MULTIPROCESSOR SYSTEM", concurrently filed herewith.

BACKGROUND OF THE INVENTION

1. Technical Field of the Invention

The present invention relates to computer systems and, in particular, to a method and apparatus for distributing interrupts in a scalable symmetric multiprocessor system.

2. Description of Related Art

The emergence of symmetric multiprocessing ("SMP") systems in today's high-end personal computer ("PC") and server markets has generated a need for new design approaches that achieve optimal performance within this expanded system structure. Some of the most significant challenges of multiprocessor system development include the design of a multiprocessor-capable bus ("MP bus") and the channeling and processing of interrupts through an SMP-aware interrupt controller. As is well understood in the art, the MP bus services multiple processing units, providing access to the main memory and other components of the system.

Conventionally, a multiprocessing system is a computer system that has more than one processor, and that is typically designed for high-end workstations or file server usage. Such a system may include a high-performance bus, huge quantities of error-correcting memory, redundant array of inexpensive disk ("RAID") drive systems, advanced system architectures that reduce bottlenecks, and redundant features such as multiple power supplies.

In the most general sense, multiprocessing is defined as the use of multiple processors to perform computing tasks. The term could apply to a set of networked computers in different locations, or to a single system containing several processors. As is well-known, however, the term is most often used to describe an architecture where two or more linked processors are contained in a single enclosure. Further, multiprocessing does not occur just because multiple processors are present. For example, having a stack of PCS in a rack is not multiprocessing. Similarly, a server with one or more "standby" processors is not multiprocessing, either. The term "multiprocessing", therefore, applies only when two or more processors are working in a cooperative fashion on a task or set of tasks.

There are many variations on the basic theme of multiprocessing. In general, the differences are related to how independently the various processors operate and how the workload among these processors is distributed. In loosely-coupled multiprocessing, the processors perform related

tasks, but, they do so as if they were standalone processors. Each processor may have its own memory and may even have its own mass storage. Further, each processor typically runs its own copy of an operating system, and communicates with the other processor or processors through a message-passing scheme, much like devices communicating over a local-area network. Loosely-coupled multiprocessing has been widely used in mainframes and minicomputers, but the software to do it is very closely tied to the hardware design. For this reason, it has not gained the support of software vendors, and is not widely used in PC servers.

In tightly-coupled multiprocessing, by contrast, the operations of the processors are more closely integrated. They typically share memory, and may even have a shared cache. The processors may not be identical to each other, and may or may not perform similar tasks. However, they typically share other system resources such as mass storage and input/output ("I/O"). Moreover, instead of a separate copy of the operating system for each processor, they typically run a single copy, with the operating system handling the coordination of tasks between the processors. The sharing of system resources makes tightly-coupled multiprocessing less expensive, and it is the dominant multiprocessor architecture in network servers.

Hardware architectures for tightly-coupled multiprocessing systems can be further divided into two broad categories. In symmetrical multiprocessor systems, system resources such as memory and disk input/output are shared by all the microprocessors in the system. The workload is distributed evenly to available processors so that one does not sit idle while another is loaded with a specific task. The performance of SMP systems increases, at least theoretically, for all tasks as more processor units are added. This highly sought-after design goal is called scalability.

In asymmetrical multiprocessor systems, tasks and system resources are managed by different processor units. For example, one processor unit may handle I/O and another may handle network operating system ("NOS") tasks. It can be readily seen that asymmetrical multiprocessor systems do not balance workloads. Thus, it is quite conceivable that a processor unit handling one task can be overworked while another unit sits idle.

It can further be noted that within the category of SMP systems are two subcategories, based on the way cache memory is implemented. The lower-performance subcategory includes "shared-cache" multiprocessing, and the higher-performance subcategory encompasses what is known as "dedicated-cache" multiprocessing. In dedicated-cache multiprocessing, every processor has, in addition to its "level 1" on-chip memory, a dedicated "level 2" off-chip memory cache (one per processor). These caches accelerate the processor-memory interaction in an MP environment. On the other hand, in shared-cache multiprocessing, the processors share a single "level 2" cache. Typically, shared-cache architecture offers less scalability than dedicated-cache architecture.

As briefly mentioned above, one of the most significant design challenges, in either broad category of multiprocessing, is the routing and processing of interrupts. Conventionally, an interrupt controller is responsible for delivering interrupts from interrupt sources to interrupt destinations in an MP system. An interrupt may be generalized as an event that indicates that a certain condition exists somewhere in the system that requires the attention of at least one processor. The action taken by a processor in response to an interrupt is commonly referred to as "servicing" or "handling" the interrupt.

In an SMP system, each interrupt has an identity that distinguishes it from the others. This identity is commonly referred to as the "vector" of the interrupt. The vector allows the servicing processor or processors to find the appropriate handler for the interrupt. When a processor accepts an interrupt, it uses the vector to locate the entry point of the handler in its interrupt table. In addition, each interrupt may have an interrupt priority that determines the timeliness with which the interrupt should be serviced relative to the other pending activities or tasks of the servicing processor.

There are, in general, two interrupt distribution modes available for an interrupt controller for delivering interrupts to their appropriate destinations in an MP system. In the directed delivery mode ("static" delivery), the interrupt is unconditionally delivered to a specific processor that matches the destination information supplied with the interrupt. Under the distributed delivery mode ("dynamic" delivery), interrupt events from a particular source will be distributed among a group of processors specified by the destination field value.

From the foregoing, it should be appreciated that balancing the interrupt loading among various processors in a scalable MP system is a very desirable goal. However, as is well understood in the art, it is a hard goal to accomplish from the system designer's perspective. Architecturally, two types of solutions exist for delivering interrupts to their destinations in an SMP environment. In one solution, for example, a centralized interrupt controller is disposed between a host bus and system bus such as the Peripheral Component Interconnect ("PCI") bus, for receiving interrupts from their sources and routing them to their destinations in either directed or distributed mode. Further, in this type of solution, a counter of certain bit-length is typically provided with each processing unit therein. The counter size is appended to a task priority register corresponding with a particular processing unit. The contents of the task priority register and the counter size appended thereto are used to determine the overall priority level for that processing unit. The counter associated with the processing unit is incremented, usually with a wraparound option, when an I/O interrupt is dispatched to that processing unit.

The second solution provides for a distributed interrupt control scheme wherein one interrupt controller assumes global, or system-level, functions such as, for example, I/O interrupt routing, while a plurality of local interrupt controllers, each of which is associated with a corresponding processing unit, control local functions such as, for example, interprocessor interrupts. Both classes of interrupt controllers may communicate through a separate bus, and are collectively responsible for delivering interrupts from interrupt sources to interrupt destinations throughout the system.

Both types of solutions described above are known to have several drawbacks. For example, in the centralized interrupt controller scheme of the foregoing, the width of the counter depends on the maximum number of processors allowed in the system, and because the width is appended to the task priority register of a processor, there is no guarantee that the selected processor for interrupt delivery in fact has the lowest priority, that is, it is the least busy unit, among the listed processors. In addition, since the scheme requires coupling of the centralized interrupt controller to the host bus and system bus, the interrupt messages will consume precious bandwidth on both buses, thereby negatively impacting the overall system performance. Further, the scalability of the centralized scheme will be degraded as more processors are added to the system.

On the other hand, although the distributed architecture has certain advantages, current distributed interrupt controller solutions also do not guarantee that the selected processor is indeed the one with the lowest priority as it is typically required that only those local interrupt controllers that have vacant interrupt slots be included in the lowest priority arbitration. Thus, it is possible that the selected processor might have the highest priority but is the only one that has at least one interrupt slot available.

A distributed interrupt controller solution that overcomes these and other disadvantages is described in greater detail in the co-assigned U.S. patent application, entitled "METHOD AND APPARATUS FOR DISTRIBUTING INTERRUPTS IN A SYMMETRIC MULTIPROCESSOR SYSTEM", cross-referenced hereinabove and concurrently being filed herewith. However, this cross-referenced patent application does not describe bus arbitration for a scalable scheme wherein multiple processors, typically in multiple sets of four processors—an architecture that is favored by digital design—are supported in a distributed interrupt controller system without changing the bus width or bus protocol.

Accordingly, it can be readily appreciated that there is a need for a cost-effective solution providing a scalable MP-compatible interrupt controller scheme that guarantees balanced delivery of interrupts to a processor that has the lowest current task priority among four or more processors without changing the bus width or bus protocol. Further, it would be advantageous to have such a scheme that is compatible with current industry architectures so as to maximize interoperability and interexchangeability. The present invention, described and claimed hereinbelow, provides a method and apparatus for accomplishing these and other objects.

SUMMARY OF THE INVENTION

A method for supporting multiple distributed interrupt controllers, designated as bus agents, in a symmetric multiprocessing system, which method includes the steps of assigning a unique identification number to each bus agent, receiving bus requests from the bus agents over four data lines in groups of four, and granting bus ownership to a selected one of the requesting bus agents.

A computer system that supports multiple distributed interrupt controllers, designated as bus agents, in a symmetric multiprocessing system, which computer system includes structure for assigning a unique identification number to each bus agent, four data lines for receiving bus requests from the bus agents in groups of four, and structure for granting bus ownership to a selected one of the requesting bus agents.

In one aspect of a presently preferred exemplary embodiment, the present invention provides a method of arbitrating bus control requests in a computer system of the type including one or more processing units, each of which is in communication with a corresponding bus agent, and a master arbiter with an internal arbitration pointer; the bus agent and the master arbiter being disposed on a programmable-interrupt-controller bus, the method comprising the steps of: initializing the contents of the internal arbitration pointer; asserting a bus control request signal by each bus agent, said asserting step being effectuated in groups of four if more than four bus agents are listed; and choosing a bus agent by the master arbiter based on an arbitration protocol, transmitting a bus message by the chosen bus agent, and updating the internal arbitration pointer to point to the chosen bus agent.

In another aspect, the present invention provides a computer system of the type including one or more processing

units, each of which is in communication with a cache memory unit, the computer system comprising one or more local programmable interrupt controllers, each of which local programmable interrupt controllers is disposed on a programmable-interrupt-controller bus, and at least one central programmable interrupt controller that is also disposed on the programmable-interrupt-controller bus, Each of the local programmable interrupt controllers comprises a current-task-priority register, an interprocessor-interrupt-command port, a who-am-i register, an interrupt-acknowledge register, an end-of-interrupt register, a first local timer, and a second local timer. Further, the central programmable interrupt controller comprises at least one feature-reporting register, at least one global-configuration register, at least one vendor-specific register, a vendor-identification register, a processor-initialization register, at least one interprocessor-interrupt-vector-priority register, a spurious-vector register, at least one global-timer register, and at least one interrupt source register. The exemplary computer system of the present invention also includes a host bus, the host bus being disposed among the cache memory units for providing a communication path therebetween; a first system bus, the first system bus being coupled to the host bus through a first bus-to-bus bridge; and a second system bus, the second system bus being coupled to the first system bus through a second bus-to-bus bridge. Also, a standard 8259A-compatible interrupt controller may be provided on the second system bus such that start-up power-on compatibility is included. Additionally, the programmable-interrupt-controller bus comprises six electrically conductive transmission lines, one clock, one control and four data lines. In a further embodiment, each of the local programmable interrupt controllers may be integrated with its corresponding processing unit.

In a yet another aspect, the present invention includes a method of delivering interrupts in a scalable multiprocessor computer system of the type including a master arbiter residing in a central interrupt controller and one or more bus agents, the central interrupt controller and the bus agents being disposed on a programmable-interrupt-controller bus, the method comprising the steps of: determining the presence of an interrupt; gaining control of the programmable-interrupt-controller bus; determining status as to whether the interrupt is a directed delivery interrupt; delivering the interrupt to a pre-specified processor if the interrupt is determined to be a directed delivery interrupt in response to the step of determining status; otherwise, selecting the unit having the lowest value in its current-task-priority register; and delivering the interrupt thereto in response to the selecting step.

In a further embodiment, the gaining control step further comprises the steps of: asserting a bus request signal by a bus agent; choosing the bus agent by the master arbiter wherein the bus agent is pointed to by an internal arbitration pointer; transmitting a bus message by the chosen bus agent; and incrementing the contents of the arbitration pointer with wraparound.

In a yet another embodiment, the selecting step further comprises the steps of: sending a distributed-interrupt command to the listed bus agents by the master arbiter; transmitting the current task priority level data serially on the programmable-interrupt-controller bus by the listed bus agents in response to the distributed-interrupt command; and determining which listed bus agent has the lowest current task priority level. In this embodiment of the invention, also: if only one bus agent has the lowest current task priority level, then choosing that bus agent for delivery of the interrupt thereto; otherwise, choosing the bus agent that has previously serviced the interrupt for interrupt delivery. Still further in this embodiment of the invention, if no agent has previously serviced the interrupt, then the one with lowest bus identification value is chosen for delivery.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete understanding of the method and apparatus of the present invention may be had by reference to the following Detailed Description when taken in conjunction with the accompanying Drawings wherein:

FIG. **1** is a block diagram of a conventional multiprocessor computer system having a centralized interrupt controller;

FIG. **2** depicts a block diagram of a second embodiment of the conventional multiprocessor computer system having a centralized interrupt controller;

FIG. **3** depicts a block diagram of a first embodiment of the present invention directed to a scalable multiprocessor computer system having a distributed interrupt control scheme wherein four or more processing units are disposed;

FIG. **4** depicts a block diagram of a second embodiment of the present invention;

FIG. **5** illustrates an exemplary flowchart for delivering interrupts in a multiprocessor computer system having a distributed interrupt control scheme in accordance with the teachings of the present invention;

FIG. **6** illustrates an exemplary flowchart for bus arbitration in a scalable multiprocessor computer system including four or more processing units;

FIG. **7** depicts an exemplary timing diagram for bus arbitration with multiple arbitration cycles in accordance with the teachings of the present invention;

FIG. **8** illustrates an exemplary embodiment of a local programmable interrupt controller in accordance with the teachings of the present invention; and

FIG. **9** illustrates an exemplary embodiment of a central programmable interrupt controller in accordance with the teachings of the present invention.

DETAILED DESCRIPTION OF DRAWINGS

Referring now to the drawings wherein like or similar elements are designated with identical reference numerals throughout the several views, and wherein the various elements depicted are not necessarily drawn to scale, and in particular to FIG. **1**, there is depicted a block diagram of a conventional multiprocessor computer system having a centralized interrupt controller **112**. The multiprocessor computer system comprises a plurality of processing units, of which two processors are shown, labeled CPU1 and CPU2 and designated with reference numerals **105** and **106**, respectively.

Each of the processors, **105** and **106**, is coupled to a corresponding cache memory unit, designated with reference numerals **107** and **108**, respectively. A system memory **109** is in communication with the processor/cache combination units via a host bus **110**. The centralized interrupt controller (labeled "CIC") **112** is connected to the host bus **110** and a first system bus **113**, such as a peripheral component interconnect bus. The first system bus **113** is bridged to the host bus **110** via a first bus-to-bus bridge, labeled PCI bridge and designated with reference numeral **111**.

Continuing to refer to FIG. **1**, the first system bus **113** is also bridged to a second system bus **115**, such as an industry

standard architecture ("ISA") bus, via a second bus-to-bus bridge, labeled PCI/ISA bridge and designated with reference numeral **114**. Disposed on the first system bus **113** are a first plurality of I/O devices, one of which devices is referenced with reference numeral **120**, that may function as I/O interrupt sources. As shown herein, each of these I/O devices is connected to the CIC **112** via its own IRQ path, such as, for example, IRQ line **121**.

Still continuing to refer to FIG. **1**, disposed on the second system bus **115** are a second I/O block **116** and a conventional 8259 interrupt controller **117**. The second I/O block **116** may also function as an interrupt source to the multiprocessor computer system. The output from the 8259 interrupt controller **117** is connected via an IRQ line **119** to the CIC **112** to enable the pass-through mode for start-up operations.

In general operation, after power-on reset, the conventional CIC **112** will default to the 8259 pass-through mode. In this mode, the 8259 interrupt request output will be passed directly through the CIC **112** to a single, pre-selected processor's interrupt request input line and the CIC **112** will be essentially disabled. During SMP operation, the 8259 pass-through mode will be disabled, and the CIC **112** will distribute all system interrupt events, as described immediately hereinbelow.

Each processor, for example CPU1 **105**, is provided in a 4-processor exemplary implementation with a two-bit counter (not shown) that is initialized to either 00, 01, 10, or 11. Each of these two-bit counters is appended to a four-bit task priority register (not shown) associated with a processor, for example CPU1 **105**. Therefore, each processor essentially has a six-bit internal priority level, and as can be readily seen, even if all four-bit task priority registers contain the same priority data, each of the four processors will have a different six-bit internal priority level. When an I/O interrupt is dispatched to any of the processors, each processor's two-bit counter is incremented by one (or wrapped around, if necessary). This causes the processor that had the lowest six-bit internal priority level before its counter is incremented to not be the lowest in priority after the increment operation.

As mentioned hereinabove, this implementation does not guarantee that the selected processor will have the lowest four-bit priority level. In addition, since this system requires that the CIC **112** be attached to both the host bus **110** and the first system bus **113**, the interrupt messages will congest and consume bus bandwidth on both buses. As can be readily appreciated, by increasing the number of processors in the system, the bus traffic for interrupt control will only exacerbate the situation, thereby compromising system scalability.

Referring now to FIG. **2**, therein is shown a block diagram of a second embodiment of the conventional multiprocessor computer system having a centralized interrupt controller, described immediately above in reference to FIG. **1**. Essentially, the embodiment in FIG. **2** can be seen to be very similar to the embodiment depicted in FIG. **1**. In the embodiment of FIG. **2**, however, the CIC **112** is integrated into a system logic chipset **205** such that it has internal connection capability to both the host bus **110** and the first system bus **113**. Although this embodiment may provide certain advantages over the embodiment of FIG. **1**, such as, for example, lower cost and pin count, it is still burdened with the disadvantages, such as, for example, consumption and congestion of the host/system buses; uncertainness in the priority of the selected processor; and less-than-optimal scalability, discussed hereinahove.

FIG. **3** depicts a block diagram of a first exemplary embodiment of the present invention directed to a scalable multiprocessor computer system having a distributed interrupt control scheme. The exemplary multiprocessor computer system may have one or more processors, two of which are labeled and designated as CPU **105** and CPU **106**. Each of the processors, for example, CPU **105** or CPU **106**, is coupled to a corresponding local programmable interrupt controller, for example, LOPIC **305** or LOPIC **306**, respectively. According to the teachings of the present invention, each LOPIC handles interrupt delivery protocol with its corresponding processor. The LOPIC also handles accesses to the processor's internal registers, inter-processor interrupts ("IPI") and remote accesses. In addition, each LOPIC can be disabled by hardware, or software. Each LOPIC may be used in conjunction with a standard 8259A-compatible interrupt controller (not shown in this FIG.) for start-up pass-through mode capability. In a presently preferred embodiment, each LOPIC contains two local timers (not shown) which may be used by the system software to monitor system performance and/or for diagnostic purposes. Further, each LOPIC contains a interprocessor-interrupt-command port (not shown), writing to which port will cause an IPI to be sent to one or more processors; and a current-task-priority register ("CTPR") which is used for setting the current task priority of each processor. The task priority indicates the relative importance of the currently executing task. In this exemplary embodiment, priority levels from 0 to 15 are implemented.

Continuing to refer to FIG. **3**, each LOPIC, for example LOPIC **305**, further contains a who-am-i register which provides a mechanism for each processor, for example, CPU **105**, to determine the ID value of that processor. This value may be used to determine the value of the destination masks used for delivering interrupts. The LOPIC **305** also contains an interrupt-acknowledge register; and an end-of-interrupt ("EOI") register, writing a zero to which register will signal the end of processing for the interrupt currently in progress for the associated processor.

FIG. **8** depicts an exemplary embodiment of LOPIC **305** and its contents, which contents may be implemented and interconnected conventionally to allow execution of LOPIC functions in embodiments of the present invention.

Referring again to FIG. **3**, each processor, for example, CPU **105**, is in data communication with an associated cache memory unit, for example, cache **107**. There may also be secondary cache; for example, cache RAM **307** and cache RAM **308**; that is connected via a data path **309** to a main memory **313**. Cache **107** and cache **108** are electrically connected to a host bus **110** which is bridged via a first bus-to-bus bridge, bridge **320**, to a first system bus **113**, such as a PCI bus. Although not shown in this FIG., it can be appreciated that a second system bus, for example, an EISA or ISA bus may be bridged to the bus **113** via a suitable bus-to-bus bridge.

Still continuing to refer to FIG. **3**, each LOPIC, for example LOPIC **305** or LOPIC **306**, is connected via a programmable-interrupt-controller bus **311** to a central programmable interrupt controller, COPIC **312**. Although only one COPIC **312** is shown in this embodiment, it can be readily understood upon reference hereto that a plurality of central programmable interrupt controllers may be used in a system constructed according to the teachings of the present invention. In an SMP environment, multiple LOPIC and COPIC units operate together and are collectively responsible for delivering interrupts from interrupt sources to interrupt destinations throughout the system. The COPIC

312 is connected to a plurality of I/O interrupt sources (not shown), and provides functions such as I/O interrupt routing, masking of interrupts and bus arbitration. The presently preferred embodiment of COPIC 312 contains global registers such as feature-reporting registers, global-configuration registers, vendor-specific registers, a vendor-identification register, a processor-initialization register, IPI-vector/priority registers, a spurious-vector register, global-timer registers, and interrupt-source registers.

FIG. 9 depicts an exemplary COPIC 312 and its contents, which contents may be implemented and interconnected conventionally to allow execution of COPIC functions in embodiments of the present invention.

Referring again to FIG. 3, the programmable-interrupt-controller bus 311 is preferably a six-wire, multi-phase, bi-directional shared bus having four data lines, one control line, and a clock line. The bus 311 is preferably optimized at present for a four-processor environment. However, as will be seen hereinbelow in specific reference to FIGS. 5, 6 and 7, the bus arbitration protocol is amenable in accordance with the teachings of the present invention to supporting more than four processors without having to change the bus width, thereby maintaining scalability. In a system with multiple COPIC and LOPIC units, one COPIC will be a priori designated as a master/arbiter and the non-master COPICs and the LOPICs (which may be integrated with their respective processing units, as shown in this FIG.) will be treated collectively as bus agents. The general operation of bus arbitration and interrupt delivery will be discussed below in specific reference to FIGS. 5, 6, and 7.

Referring now to FIG. 4, therein is depicted a block diagram of a second embodiment of the present invention directed to a multiprocessor computer system having a distributed interrupt control scheme. This embodiment is similar to the one discussed in detail hereinabove in reference to FIG. 3. However, the LOPIC units, 305 and 306, are provided in this embodiment as external units that are in communication with their corresponding processors, CPU 105 and 106, via external paths 401 and 402, respectively, rather than as integrated units with processors having internal access thereto. It can be readily understood that the exemplary embodiment discussed in reference to FIG. 3 provides a better solution in terms of lower cost and lower pin count than the one described in this FIG., but this design is an alternative design that may be usefully implemented in some applications.

FIG. 5 is an exemplary flowchart for delivering interrupts in an SMP environment in accordance with the teachings of the present invention. During system power up and initialization step 505, each bus agent (discussed in reference to FIG. 3) is assigned a unique arbitration identification ("Arb ID") value. Because of the 8259A-compatibility, the power up reset will have been processed by a prespecified processor in the system. In addition, each bus agent will be initialized as to the total number of bus agents on the programmable-interrupt-controller bus 311 (shown in FIG. 3). The preferred method is to allow the system software ("OS") to control and provide all the necessary information during the system initialization phase 505. In order to maintain scalability, the preferred embodiment employs an encoded 5-bit register wherein a value of "00000" indicates the presence of one bus agent and one master COPIC; a value of "00001" indicates two bus agents and one master COPIC; and so on, up to a value of "11110" indicating 31 bus agents and one master COPIC. Additionally, a value of "11111" in the encoded 5-bit register indicates a RESET in the exemplary embodiment.

If the system detects the presence of an interrupt, as provided in the decision block 510, then either a COPIC or a LOPIC must gain control of the programmable-interrupt-controller bus 311 before a bus message may be delivered therethrough. If the bus 311 is busy, as determined in the decision block 515, then a bus agent must wait until the bus 311 is idle, as indicated in the decision block 517. Then, once the current bus transaction is complete, the bus agent must arbitrate to gain a bus grant, since only the master/arbiter COPIC will have total control of its bus request line which is connected to the control line of the bus. The arbitration phase 520 will be discussed in greater detail hereinbelow in specific reference to FIGS. 6 and 7, taken together. If the bus 311 is idle, an agent or the master may directly start an arbitration cycle to gain a bus grant.

Upon entering the decision block 525 after gaining control of the bus 311, a determination is made if the interrupt is a directed delivery mode interrupt or a distributed delivery mode interrupt. If the determination is that the interrupt is a directed delivery mode interrupt, then by taking the YES path therefrom, data is delivered to the destination processor (or, bus agent/LOPIC, if the processor and its LOPIC are not integrated together) on a plurality of data lines, for example, four data lines, that comprise a portion of the bus 311.

If the determination in step 525 is that the interrupt is a distributed delivery mode interrupt, then a COPIC will send a "distributed interrupt" command to all listed agents, as provided in step 530. Each agent then provides its current task priority level (CTPR) by transmitting its four-bit priority level serially via the data line that is used to request the bus with. Because the number of bits in a CTPR level is four, a total of four cycles is required for each agent to provide its CTPR value to the COPIC.

After receiving the CTPR values, the COPIC compares and selects in step 535 the agent that has the is lowest CTPR value (0 is the lowest, meaning the least busy; 15 is the highest). If only one agent has the lowest CTPR value, then by taking the YES path from decision block 540, the interrupt is delivered to that agent. Otherwise, a determination is made in decision block 550 to select the agent that previously serviced the same interrupt for delivery thereof as provided in step 555. If, on the other hand, no agent previously serviced that interrupt, then a choice is made in step 565 to select the agent based on its unique Arb ID, for example, selecting the one with the lowest Arb ID for delivering the interrupt thereto.

Referring now to FIGS. 6 and 7 together, a scheme for bus arbitration in an exemplary scalable MP system having four or more processing units (each of which may be integrated with its respective LOPIC) is described in a flowchart in FIG. 6 and as a bus timing diagram in FIG. 7. As can be readily appreciated, the same bus arbitration scheme may be used in an embodiment having less than four processing units without deviating from the scope of the present invention. For example, if there are only two processing units, only two data lines will be pulled HIGH for arbitration, as will be described hereinbelow in reference to a exemplary generic embodiment.

The arbitration phase, which was designated as step 520 in FIG. 5, is entered into with the start of a bus transaction shown in the step 601. After the initialization step 602, during the first arbitration cycle, the first four bus agents issue their respective bus request signals by asserting ACTIVE HIGH on their own bus request lines that are connected to the data lines of the bus 311 (shown in FIG. 3). For example, during the first arbitration cycle, agent 0 which

is connected to OPDATA 0 (shown in FIG. 7), drives that line, referred to as DATA 0 in this flowchart. Similarly, agent 1 which is connected to OPDATA 1 (shown in FIG. 7), drives that line, referred to as DATA 1 in this flowchart; agent 2 which is connected to OPDATA 2 (shown in FIG. 7), drives that line, referred to as DATA 2 in this flowchart; and agent 3 which is connected to OPDATA 3 (shown in FIG. 7), drives that line, referred to as DATA 3 in this flowchart. If no more agents are present or detected, as provided in the step **604**, the master/arbiter COPIC will select a bus agent by employing a "rotating priority" or "round robin" arbitration protocol to grant the bus **311** for message delivery, as shown in steps **607** and **608**. For this purpose, the master/arbiter COPIC in the exemplary embodiment utilizes an internal circular pointer, denoted as ARB_PTR, that points to the bus agent that has been granted the most recent bus request. According to one aspect of the present invention, the bus agent that is pointed to by the contents of the ARB_PTR is accorded the least priority of bus arbitration; and, further, the bus agent that is pointed to by a wraparound increment of the ARB_PTR value by one is accorded the highest priority. In a presently preferred exemplary embodiment, a 5-bit ARB_PTR is provided for scalably accommodating up to 32 units, 31 bus agents and one master COPIC. On power-up, this 5-bit ARB_PTR is initialized to "11111" such that the bus agent pointed to by "00000" will have the highest priority for the up-coming bus arbitration. If, on the other hand, the total number of bus agents and master COPIC is less than 32, then, the most significant bits of the ARB_PTR will be appropriately masked.

Continuing to refer to FIG. **6**, once the control of the bus **311** is granted, the requester may immediately start sending its message, with a Cyclic Redundancy Check ("CRC") and ERROR check (shown in FIG. **7**). After each successfully transmitted message, the master/arbiter COPIC increments or updates its ARB_PTR to point to the bus agent that successfully transmitted the message, as provided in step **609**. At the end of current bus transaction, the bus arbitration process phase may begin again for the next bus transaction **610**.

On the other hand, if more than four agents are arbitrating for bus control, then by taking the NO path from the decision block **604**, the arbitration cycle counter is incremented, as in step **606**, and the next arbitration cycle may start over again, with additional agents, for example, agents **4**, **5**, **6** and **7** driving data lines OPDATA **0**, OPDATA **1**, OPDATA **2**, and OPDATA **3**, respectively, as shown in FIG. **7**. If the list of agents is exhausted, as provided in **604**, then the bus control is granted in the same manner as discussed above.

From the foregoing, it can be readily appreciated by those skilled in the art that the present invention provides a highly cost-effective solution for balanced delivery of interrupts to their destinations in a scalable MP environment having four or more processing units without changing the bus width or bus protocol. The disclosed method requires that only CTPR values for each bus agent be compared for determining lowest current task priority, thereby ensuring that in the dynamic delivery mode, an interrupt is always distributed to the agent that is least busy. Further, by distributing control between local interrupt controllers and central interrupt controllers that are disposed on a dedicated interrupt bus, it can be realized that the impact on the host bus traffic or system bus traffic is rendered vanishingly small. With integration of local interrupt controllers with their corresponding processing units, a low cost, low pin count solution that is highly scalable in an MP system architecture having four or more processors is achieved. Furthermore, as described hereinabove, the present invention may also be practiced with fewer than four processing units without having to change the bus width or bus protocol. Accordingly, it is to be understood that the present invention provides a highly scalable solution that can support one or more processing units in an MP-environment.

Although a preferred embodiment of the method and apparatus of the present invention has been illustrated in the accompanying Drawings and described in the foregoing Detailed Description, it will be understood that the invention is not limited to the embodiment disclosed, but is capable of numerous rearrangements, modifications and substitutions without departing from the spirit of the invention as set forth and defined by the following claims.

What is claimed is:

1. A method for supporting multiple distributed interrupt controllers, designated as bus agents in a symmetric multiprocessing system, said method comprising the steps of:

assigning a unique identification number to each bus agent;

receiving bus requests from the bus agents in groups of four over a single programmable-interrupt-controller bus comprising four data lines;

arbitrating for control of said propgrammable-interrupt-controller bus, said arbitration step involving one or more arbitration phases each one of which phases corresponds to a particular group of four agents and is effectuated on said single programmable-interrupt-controller bus; and

granting bus ownership to a selected one of the requesting bus agents.

2. The method as set forth in claim **1**, wherein up to thirty-two bus agents can be supported without changing bus protocol or width associated with said single programmable-interrupt-controller bus.

3. A computer system that supports multiple distributed interrupt controllers, designated as bus agents, in a symmetric multiprocessing system, said computer system comprising:

a single programmable-interrupt-controller bus comprising four data lines;

means for assigning a unique identification number to each bus agent;

means for receiving bus requests from the bus agents in groups of four;

means for accommodating an arbitration scheme wherein an arbitration cycle comprises one or more arbitration phases, each one of which phases corresponds to a particular group of four bus agents and is effectuated on said single programmable-interrupt-controller bus; and

means for granting bus ownership to a selected one of the requesting bus agents.

4. The computer system as set forth in claim **3**, wherein up to thirty-two bus agents can be supported without changing bus protocol or width associated with said single programmable-interrupt-controller bus.

5. The computer system as recited in claim **3**, wherein said means for assigning comprises a central processor.

6. The computer system as recited in claim **3**, wherein said means for granting comprises a central programmable interrupt controller.

7. A method of arbitrating bus control requests in a computer system of the type including one or more processing units, each of which is in communication with a corresponding bus agent; and a master arbiter with an internal

arbitration pointer; the bus agents and the master arbiter disposed on a single programmable-interrupt-controller bus, the method comprising the steps of:

initializing the contents of the internal arbitration pointer;

receiving bus requests from the bus agents in groups of four over said single programmable-interrupt-controller bus comprising four data lines;

arbitrating for control of said single programmable-interrupt-controller bus, said arbitration step involving one or more arbitration phases, each one of which phases corresponds to a particular group of four agents; and

choosing a bus agent by the master arbiter based on an arbitration protocol, transmitting a bus message by the chosen bus agent, and updating the internal arbitration pointer to point to the chosen bus agent.

8. The method of arbitrating bus control requests in a computer system as set forth in claim **7**, further comprising the steps of:

initializing an arbitration cycle if more than four bus agents are listed; and

initializing a counter if more than four bus agents are listed.

9. The method of arbitrating bus control requests in a computer system as set forth in claim **8**, wherein said arbitration cycle is initialized by setting it to one and said counter is initialized by setting it to zero.

10. A computer system including a subsystem for arbitrating bus control requests of the type including one or more processing units, each of which is in communication with a corresponding bus agent, the bus agents disposed on a single programmable-interrupt-controller bus, the subsystem comprising:

means for initializing an internal arbitration pointer;

means for asserting a bus control request signal;

means for accommodating an arbitration scheme wherein an arbitration cycle comprises one or more arbitration phases, each one of which phases corresponds to a particular group of four bus agents and is effectuated on said single programmable-interrupt-controller bus;

means for choosing a listed bus agent based on said arbitration cycle; and

means for updating said internal arbitration pointer.

11. The computer system as set forth in claim **10**, wherein said means for asserting a bus control request comprises a signal generating means for generating an active high signal.

12. The computer system as set forth in claim **10**, wherein said means for choosing comprises a central programmable interrupt controller disposed on said single programmable-interrupt-controller bus.

* * * * *

US005918057A

United States Patent [19]

Chou et al.

[11] Patent Number: 5,918,057

[45] Date of Patent: Jun. 29, 1999

[54] **METHOD AND APPARATUS FOR DISPATCHING MULTIPLE INTERRUPT REQUESTS SIMULTANEOUSLY**

[75] Inventors: **Hong-Chich Chou**, Hsinchu; **Jerng-Cherng Fan**, Hsingchu; **Tsahn-Ylh Chang**, Hsinchu; **Po-Chuan Kang**, Hsingchu, all of Taiwan

[73] Assignee: **Industrial Technology Research Institute**, Hsinchu, Taiwan

[21] Appl. No.: **08/822,191**

[22] Filed: **Mar. 20, 1997**

[51] **Int. Cl.**[6] **G06F 9/46**

[52] **U.S. Cl.** **395/733**; 395/735; 395/740; 395/672

[58] **Field of Search** 395/733–742, 395/672–675, 390–391

[56] **References Cited**

U.S. PATENT DOCUMENTS

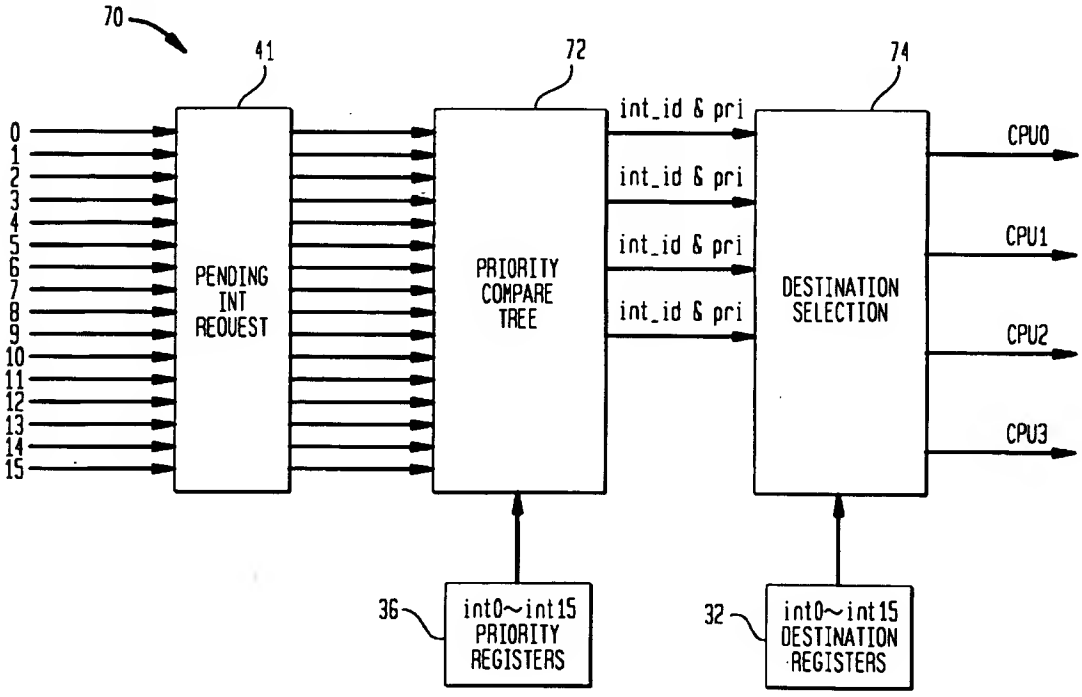
4,394,727	7/1983	Hoffman et al.	395/673
5,109,512	4/1992	Bahr et al.	395/673
5,125,093	6/1992	McFarland	395/739
5,247,628	9/1993	Grohoski	395/390
5,283,904	2/1994	Carson et al.	395/739
5,301,324	4/1994	Dewey et al.	395/675
5,452,452	9/1995	Gaetner et al.	395/673
5,515,538	5/1996	Kleiman	395/733

Primary Examiner—Ayaz R. Sheikh
Assistant Examiner—Sumati Lefkowitz
Attorney, Agent, or Firm—Proskauer Rose LLP

[57] **ABSTRACT**

An interrupt processing method and apparatus particularly well-suited for use in an interrupt controller of a multiprocessor system or device. Each of the interrupt requests has at least one destination processor associated therewith for servicing the interrupt request. An interrupt controller in accordance with the present invention applies latched interrupt requests to a priority compare tree which serves to prioritize received interrupt requests. A number of higher priority requests, including the highest priority request, are supplied to a destination selection circuit which includes an interrupt dispatcher which determines a processor to which the first priority interrupt request will be dispatched. Similar determinations are made for the remaining identified interrupt requests, but with the corresponding destination register contents masked to prevent processors already selected to receive a higher priority interrupt from being considered for a lower priority interrupt. The destination selection circuit attempts to determine a unique destination processor for each of the highest priority interrupt requests, such that these multiple interrupt requests can therefore be dispatched to different processors simultaneously. One or more of the interrupt requests may be "blocked" during a particular time period because all destination processors which could service the blocked requests are already processing other interrupts, performing higher priority tasks or are otherwise unavailable. These blocked interrupt requests are identified and the corresponding destination registers are masked such that the remaining non-blocked interrupt requests can be delivered to an available destination processor.

10 Claims, 15 Drawing Sheets



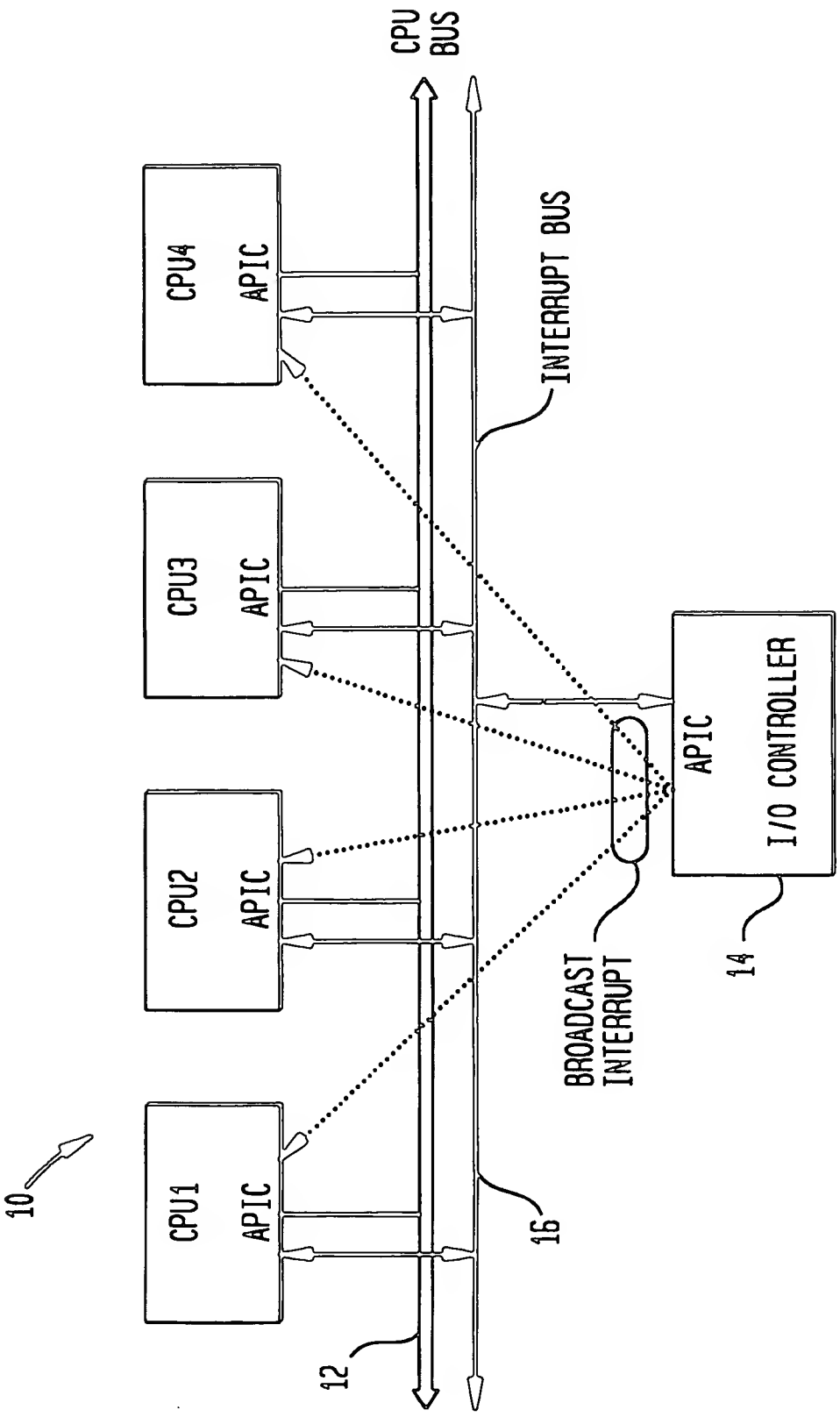


FIG. 1
(PRIOR ART)

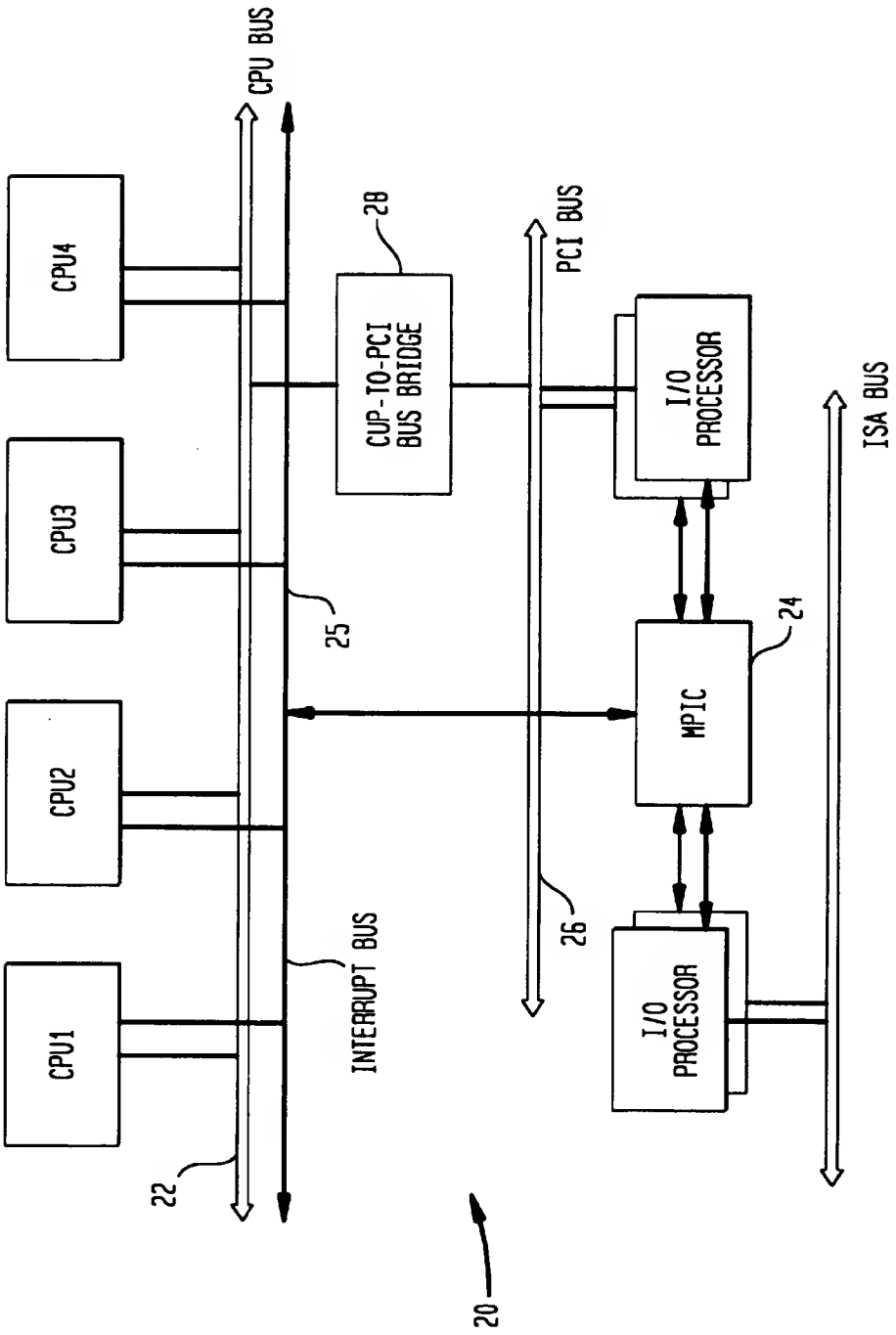


FIG. 2
(PRIOR ART)

FIG. 3

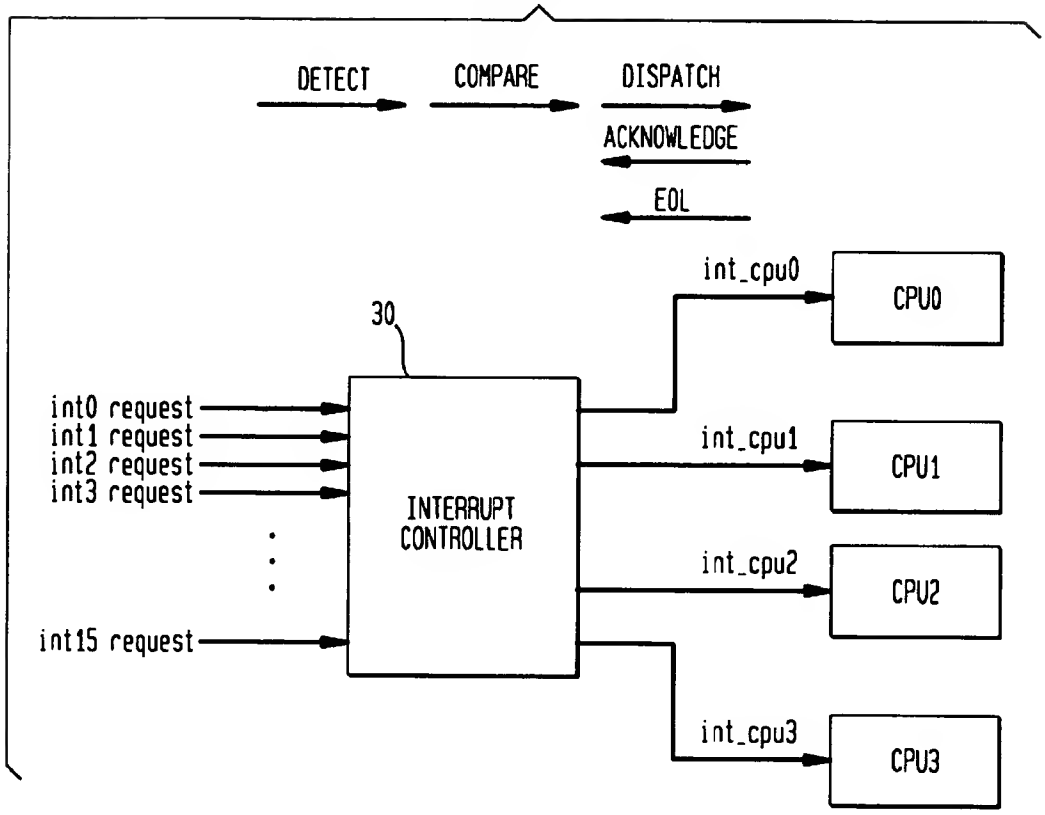

DETECT
COMPARE
DISPATCH
ACKNOWLEDGE
EOL
30
int0 request
int1 request
int2 request
int3 request
int15 request
INTERRUPT CONTROLLER
int_cpu0
int_cpu1
int_cpu2
int_cpu3
CPU0
CPU1
CPU2
CPU3

FIG. 4

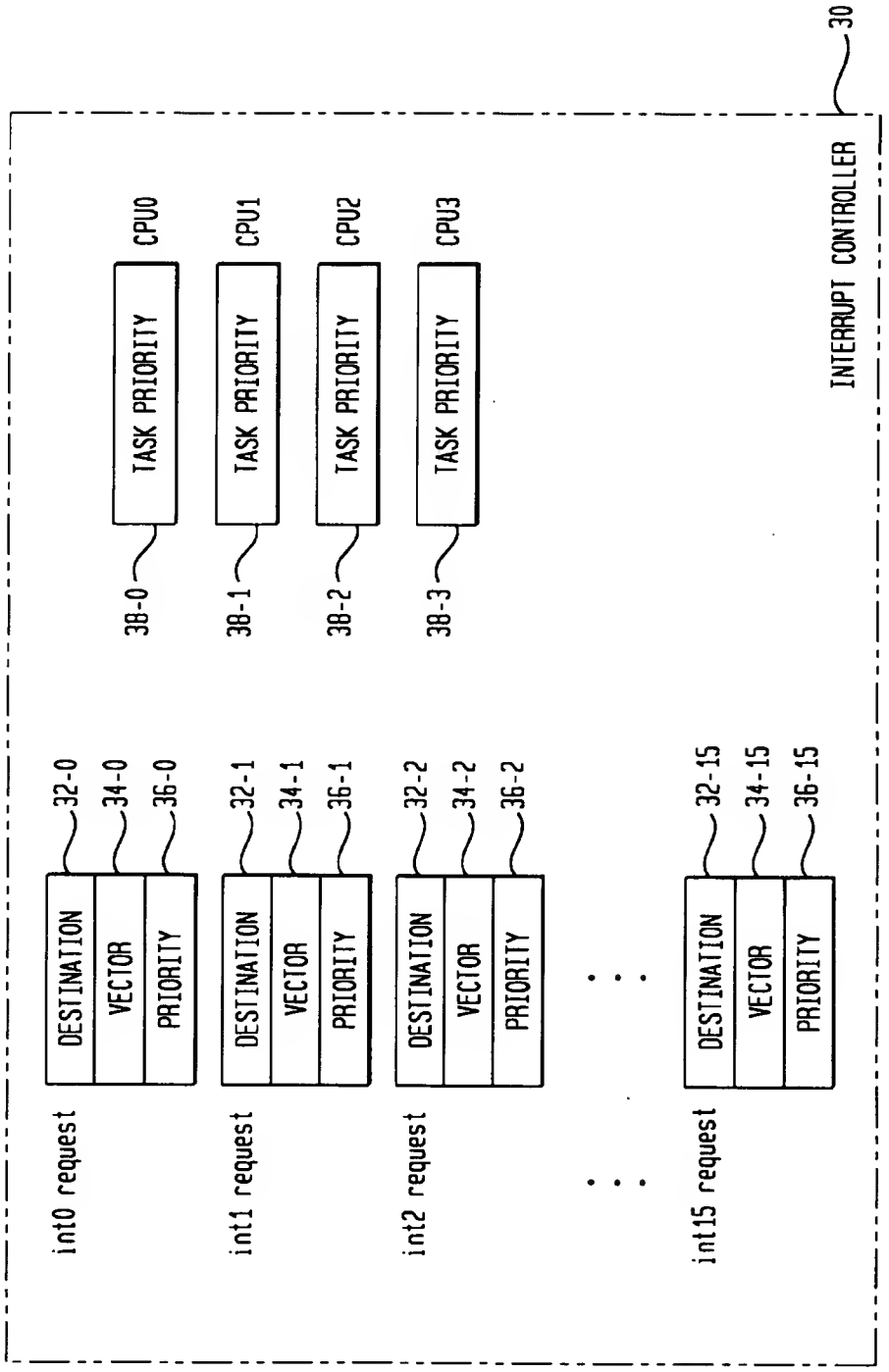
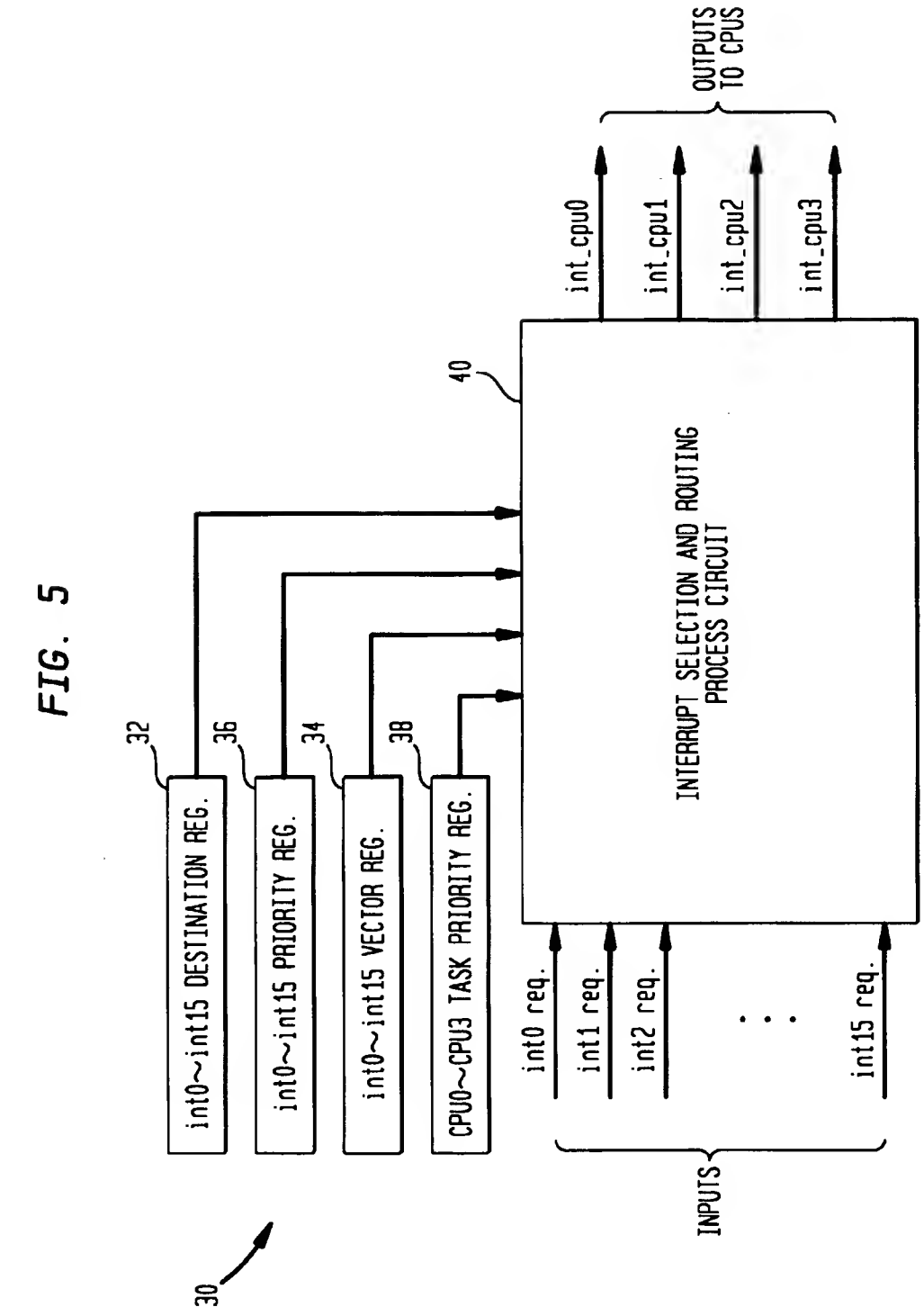

30
INTERRUPT CONTROLLER
CPU0
CPU1
CPU2
CPU3
TASK PRIORITY
38-0
38-1
38-2
38-3
32-0
34-0
36-0
32-1
34-1
36-1
32-2
34-2
36-2
32-15
34-15
36-15
DESTINATION
VECTOR
PRIORITY
int0 request
int1 request
int2 request
int15 request

FIG. 5


OUTPUTS TO CPUS
int_cpu0
int_cpu1
int_cpu2
int_cpu3
40
INTERRUPT SELECTION AND ROUTING PROCESS CIRCUIT
int0 req.
int1 req.
int2 req.
int15 req.
INPUTS
32
int0~int15 DESTINATION REG.
36
int0~int15 PRIORITY REG.
34
int0~int15 VECTOR REG.
38
CPU0~CPU3 TASK PRIORITY REG.
30

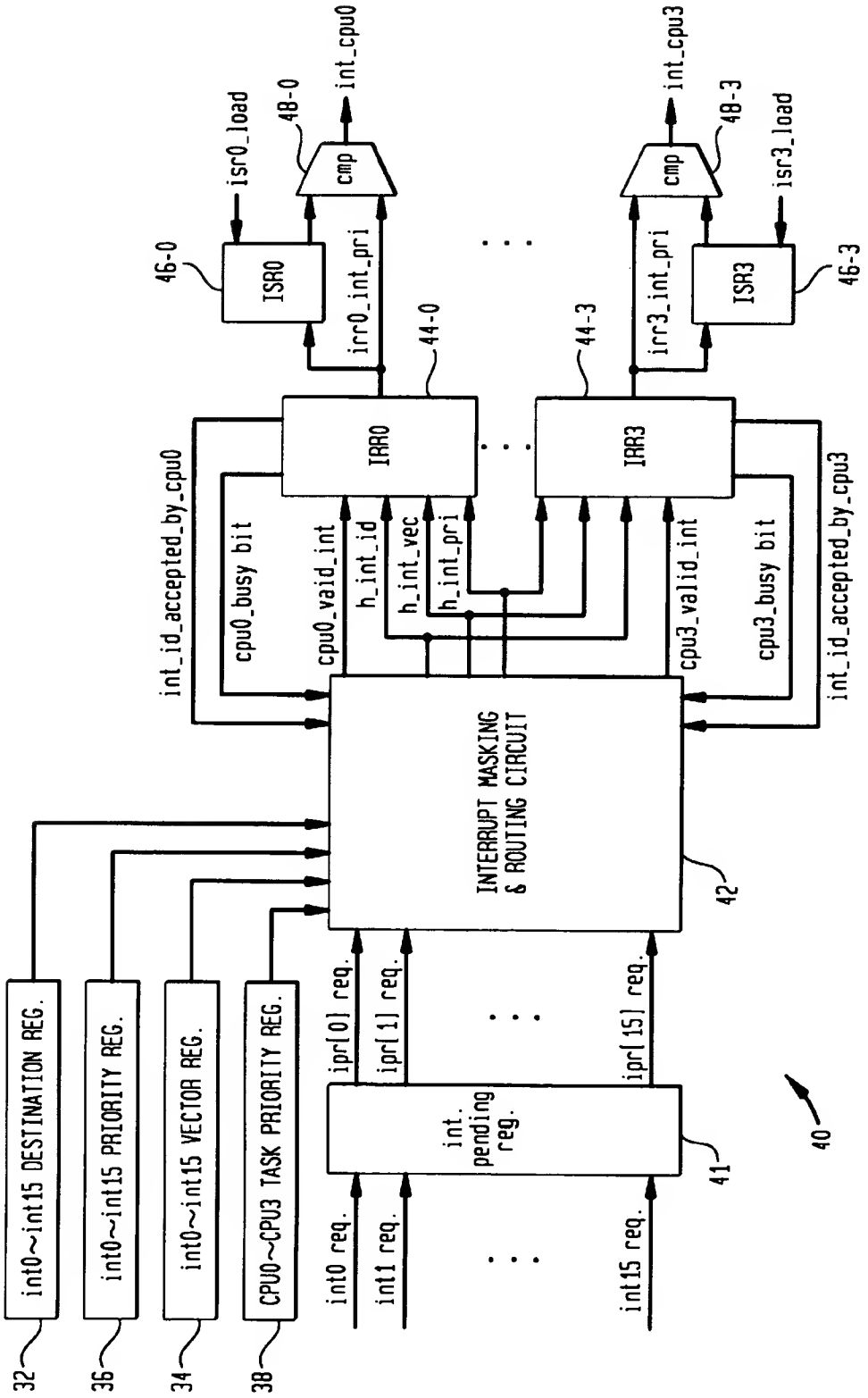

int0~int15 DESTINATION REG.
int0~int15 PRIORITY REG.
int0~int15 VECTOR REG.
CPU0~CPU3 TASK PRIORITY REG.
32
36
34
38
INTERRUPT MASKING & ROUTING CIRCUIT
42
int. pending reg.
41
40
ipr[0] req.
ipr[1] req.
ipr[15] req.
int0 req.
int1 req.
int15 req.
IRR0
IRR3
44-0
44-3
irr0_int_pri
irr3_int_pri
ISR0
ISR3
46-0
46-3
isr0_load
isr3_load
48-0
48-3
cmp
int_cpu0
int_cpu3
cpu0_busy bit
cpu0_vaid_int
h_int_id
h_int_vec
h_int_pri
cpu3_valid_int
cpu3_busy bit
int_id_accepted_by_cpu0
int_id_accepted_by_cpu3

FIG. 6

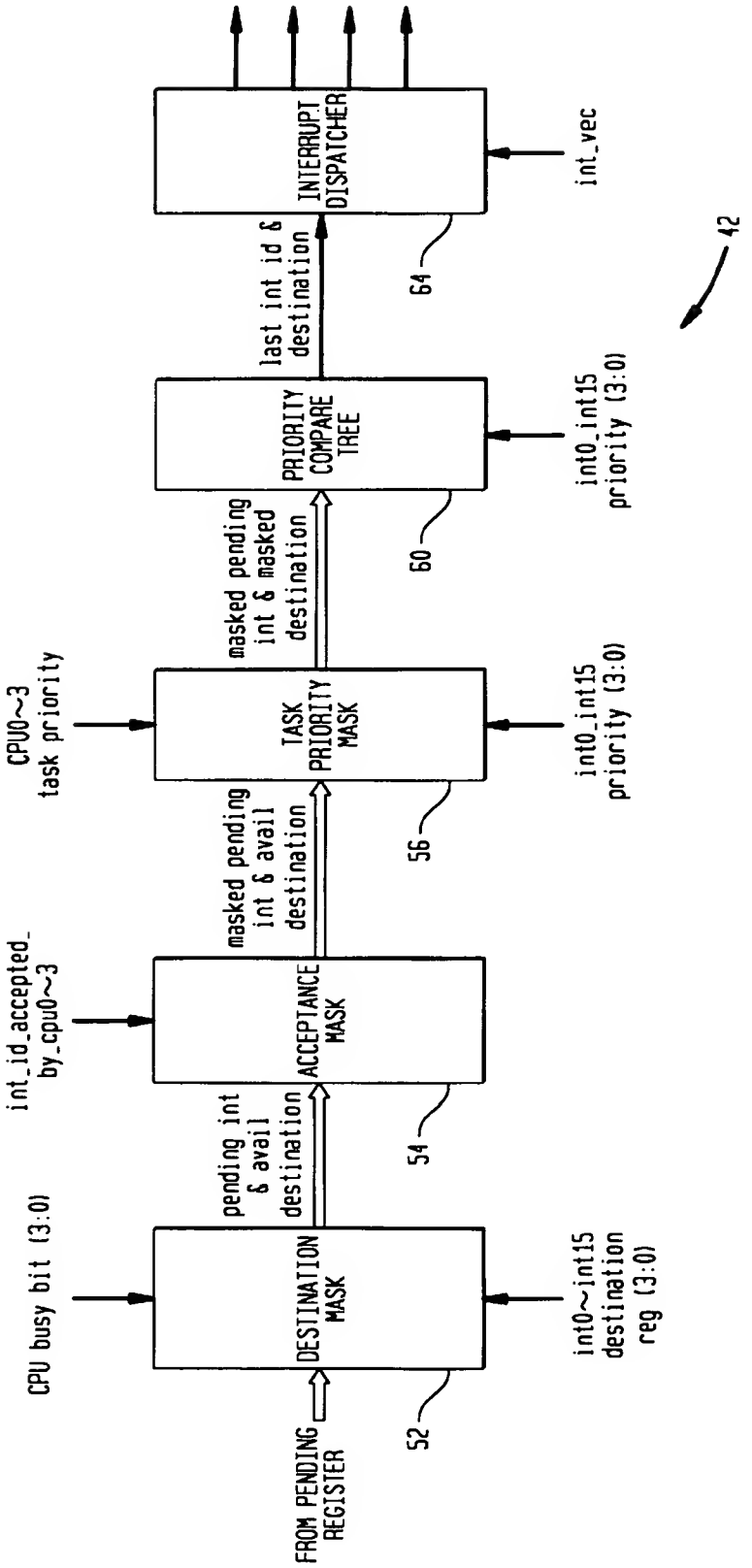

INTERRUPT DISPATCHER
int_vec
64
last int id & destination
PRIORITY COMPARE TREE
int0_int15 priority (3:0)
60
masked pending int & masked destination
TASK PRIORITY MASK
CPU0~3 task priority
int0_int15 priority (3:0)
56
masked pending int & avail destination
ACCEPTANCE MASK
int_id_accepted_ by_cpu0~3
54
pending int & avail destination
DESTINATION MASK
CPU busy bit (3:0)
int0~int15 destination reg (3:0)
52
FROM PENDING REGISTER
42

FIG. 7

FIG. 8A

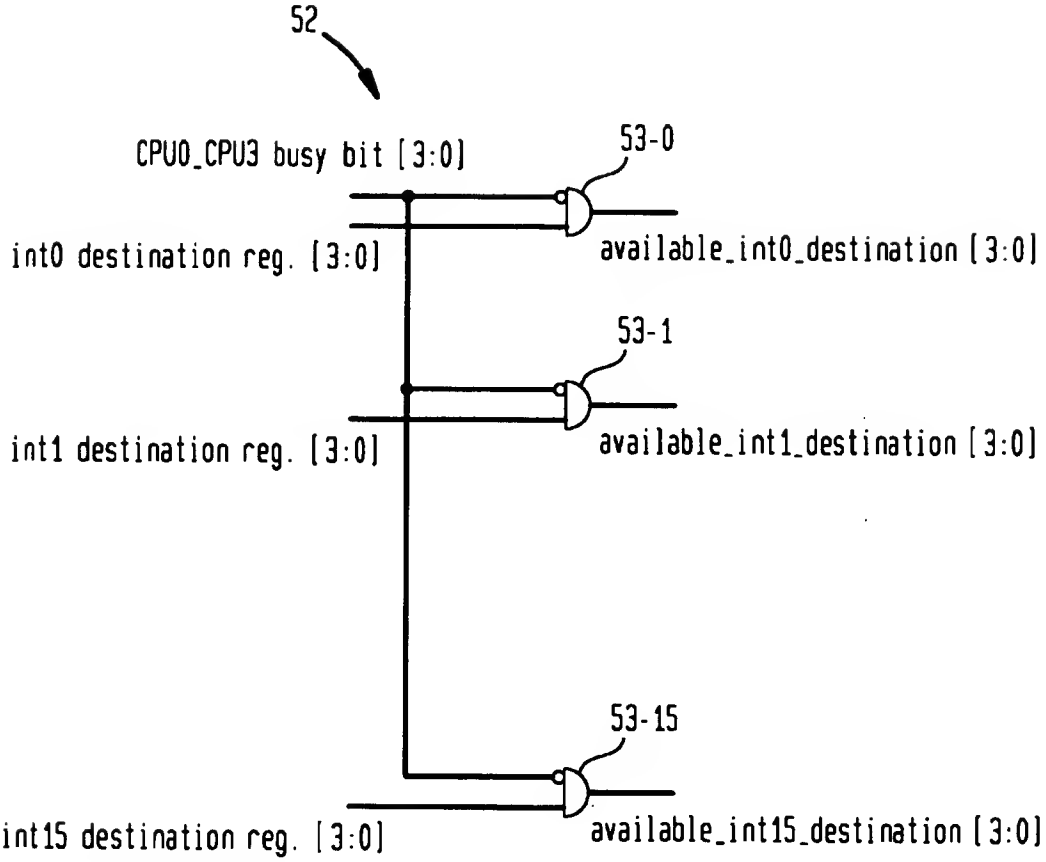
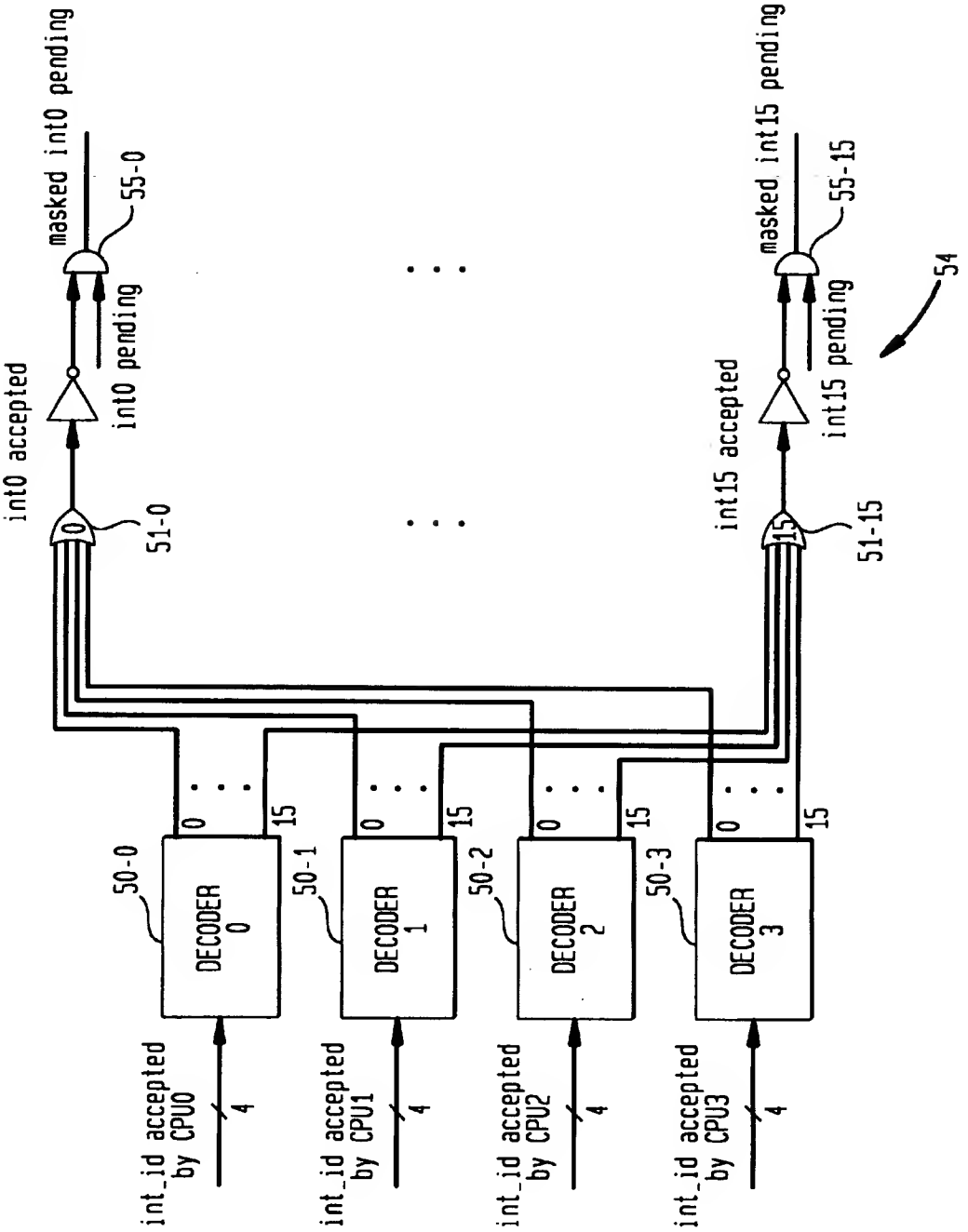

52
CPU0_CPU3 busy bit [3:0]
53-0
int0 destination reg. [3:0]
available_int0_destination [3:0]
53-1
int1 destination reg. [3:0]
available_int1_destination [3:0]
53-15
int15 destination reg. [3:0]
available_int15_destination [3:0]

FIG. 8B


masked int0 pending
55-0
int0 pending
int0 accepted
51-0
masked int15 pending
55-15
int15 pending
int15 accepted
51-15
54
DECODER 0
DECODER 1
DECODER 2
DECODER 3
50-0
50-1
50-2
50-3
int_id accepted by CPU0
int_id accepted by CPU1
int_id accepted by CPU2
int_id accepted by CPU3
4
0
15

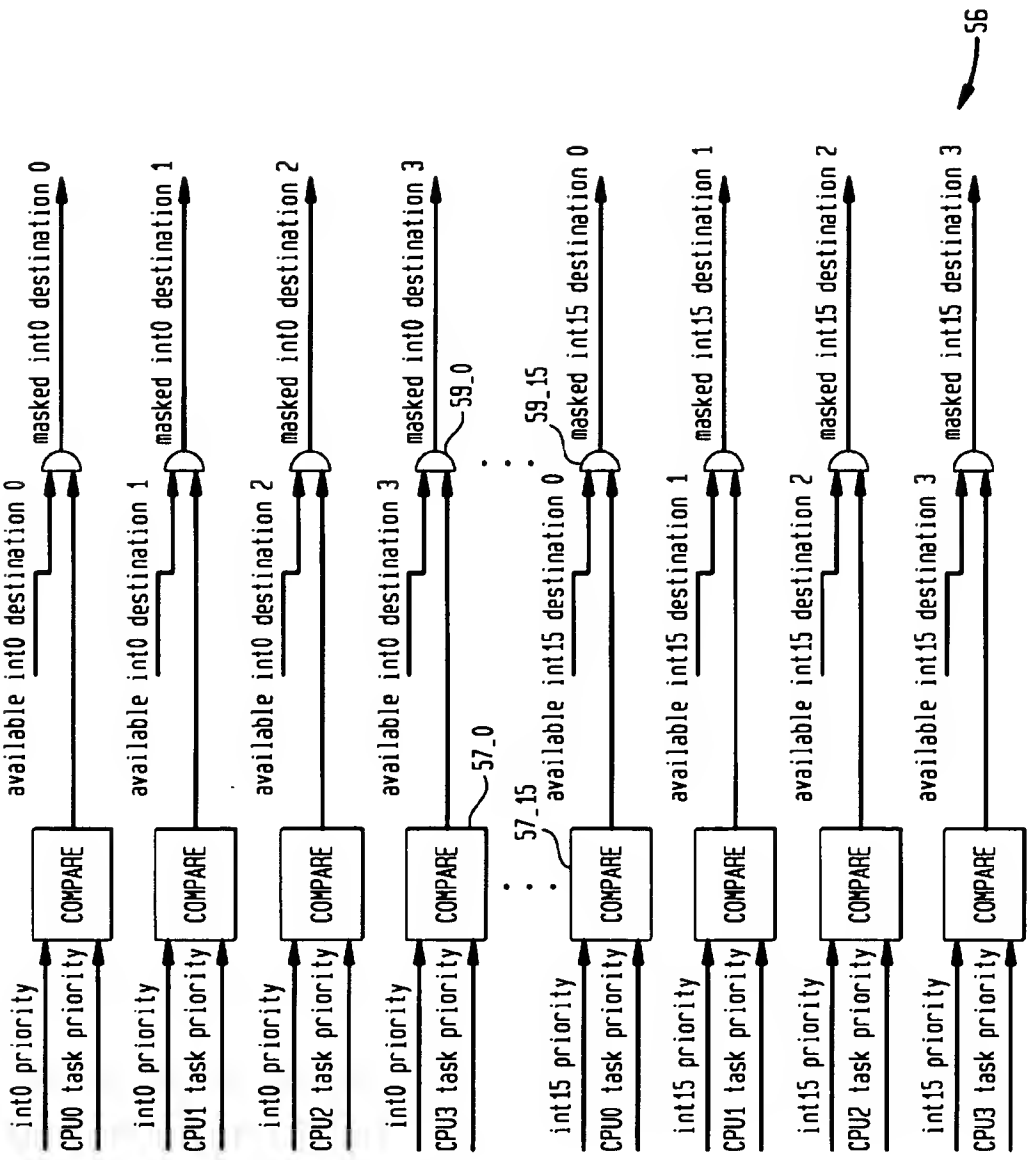

56
available int0 destination 0
masked int0 destination 0
available int0 destination 1
masked int0 destination 1
available int0 destination 2
masked int0 destination 2
available int0 destination 3
masked int0 destination 3
59_0
57_0
57_15
available int15 destination 0
masked int15 destination 0
59_15
available int15 destination 1
masked int15 destination 1
available int15 destination 2
masked int15 destination 2
available int15 destination 3
masked int15 destination 3
COMPARE
int0 priority
CPU0 task priority
CPU1 task priority
CPU2 task priority
CPU3 task priority
int15 priority

FIG. 8C

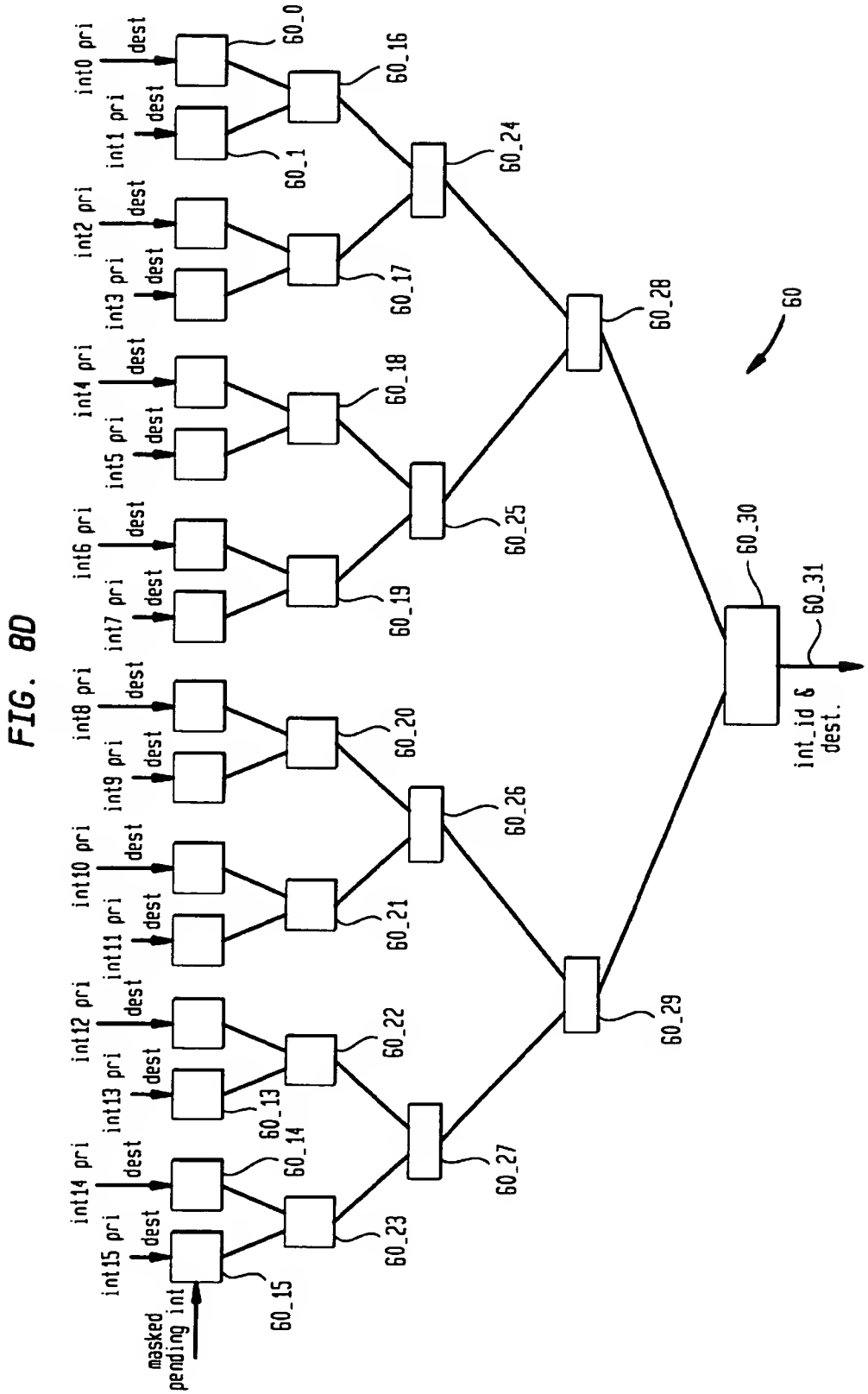

int0 pri
dest
int1 pri
dest
60_0
60_1
60_16
int2 pri
dest
int3 pri
dest
60_17
60_24
int4 pri
dest
int5 pri
dest
60_18
60_25
int6 pri
dest
int7 pri
dest
60_19
60_28
int8 pri
dest
int9 pri
dest
60_20
60_26
int10 pri
dest
int11 pri
dest
60_21
60_29
int12 pri
dest
int13 pri
dest
60_13
60_22
int14 pri
dest
int15 pri
dest
60_14
60_23
60_27
masked pending int
60_15
60_30
60_31
int_id & dest.
60

FIG. 8D

FIG. 8E

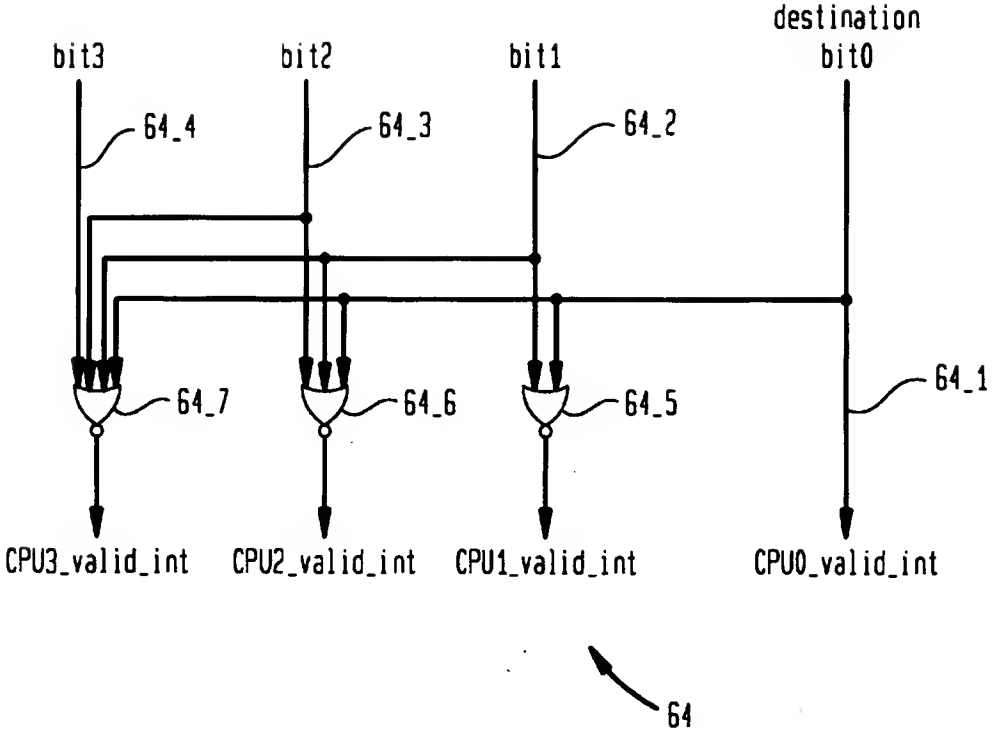
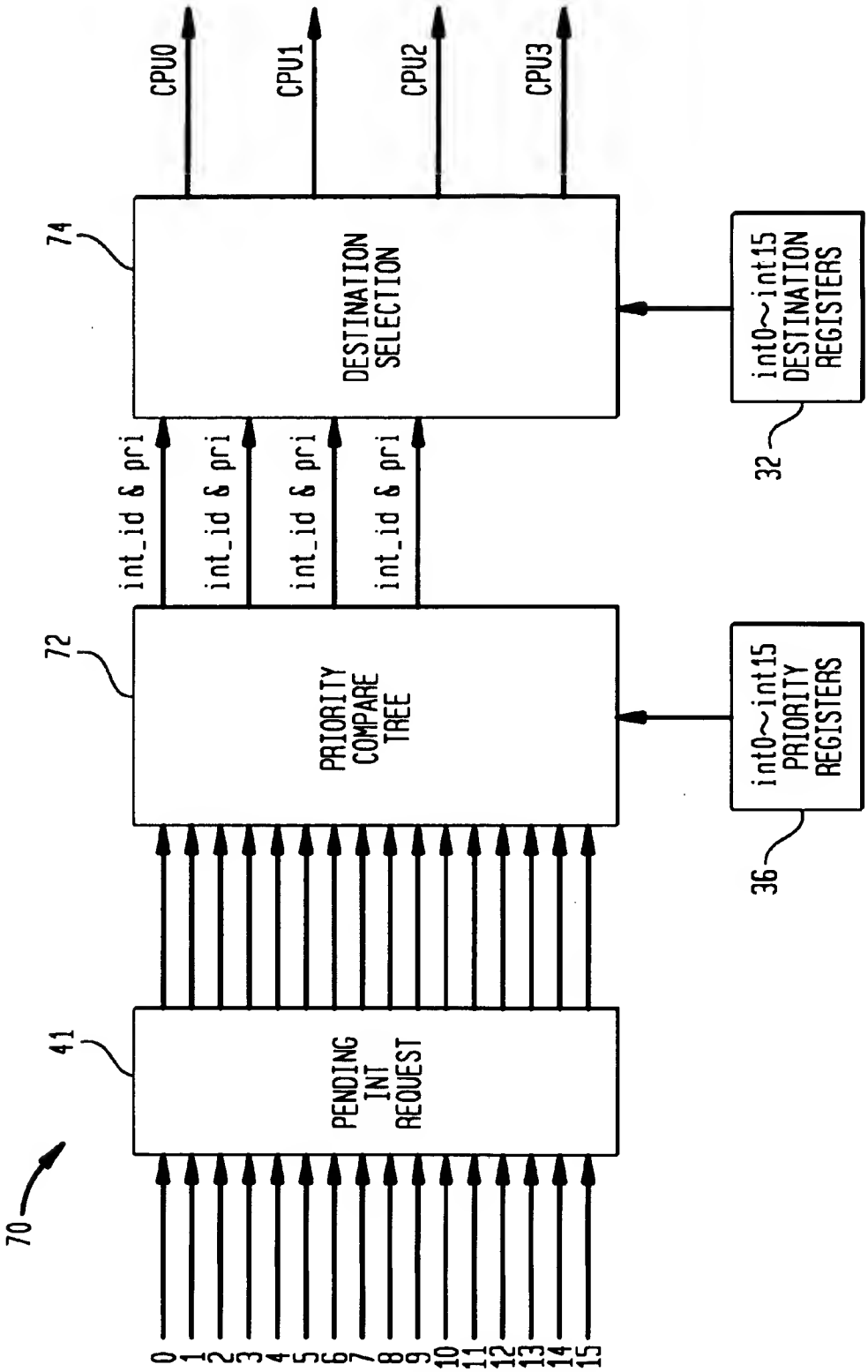

bit3
bit2
bit1
destination bit0
64_4
64_3
64_2
64_1
64_7
64_6
64_5
CPU3_valid_int
CPU2_valid_int
CPU1_valid_int
CPU0_valid_int
64

FIG. 9


CPU0
CPU1
CPU2
CPU3
74
DESTINATION SELECTION
int0~int15 DESTINATION REGISTERS
32
int_id & pri
int_id & pri
int_id & pri
int_id & pri
72
PRIORITY COMPARE TREE
int0~int15 PRIORITY REGISTERS
36
41
PENDING INT REQUEST
70
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

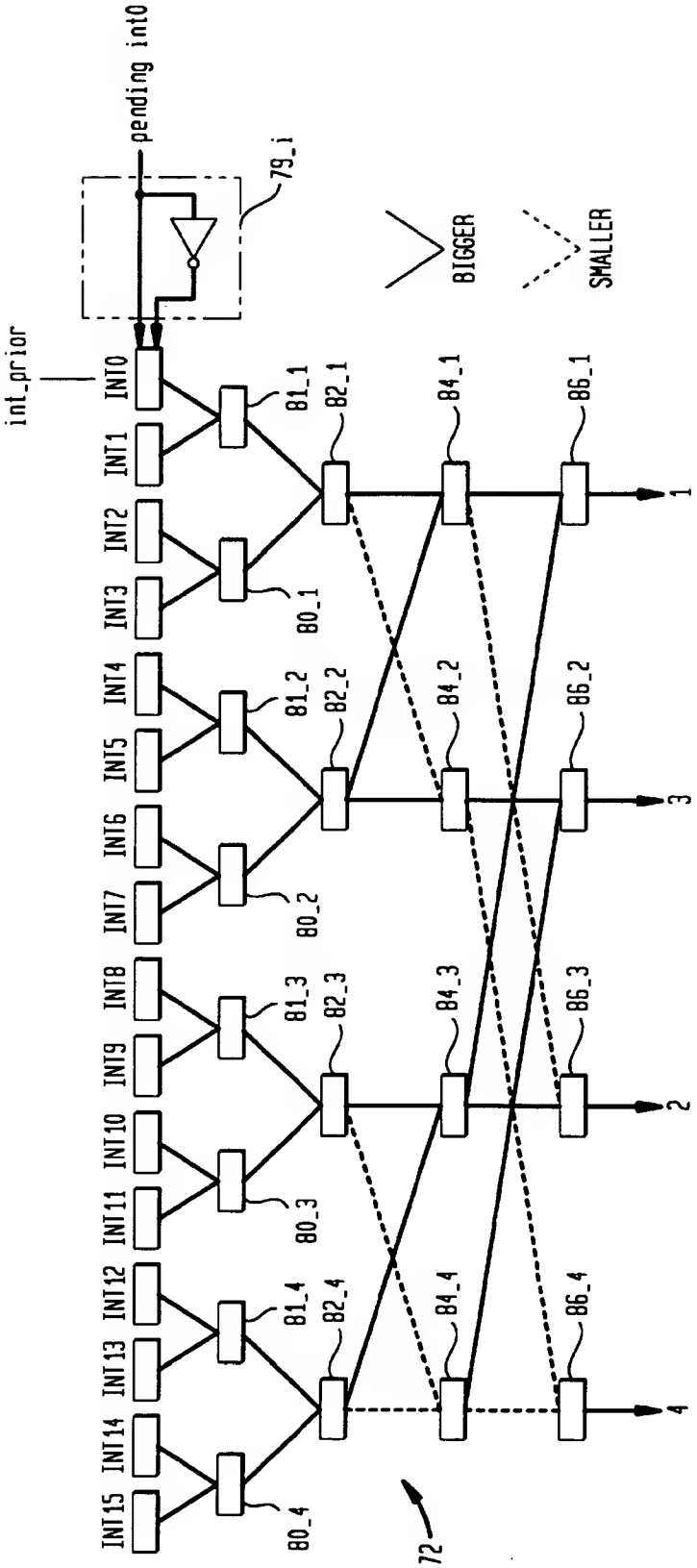

pending int0
79_i
BIGGER
SMALLER
int_prior
INT0
INT1
INT2
INT3
INT4
INT5
INT6
INT7
INT8
INT9
INT10
INT11
INT12
INT13
INT14
INT15
81_1
82_1
84_1
86_1
80_1
81_2
82_2
84_2
86_2
80_2
81_3
82_3
84_3
86_3
80_3
81_4
82_4
84_4
86_4
80_4
1
3
2
4
72

FIG. 10

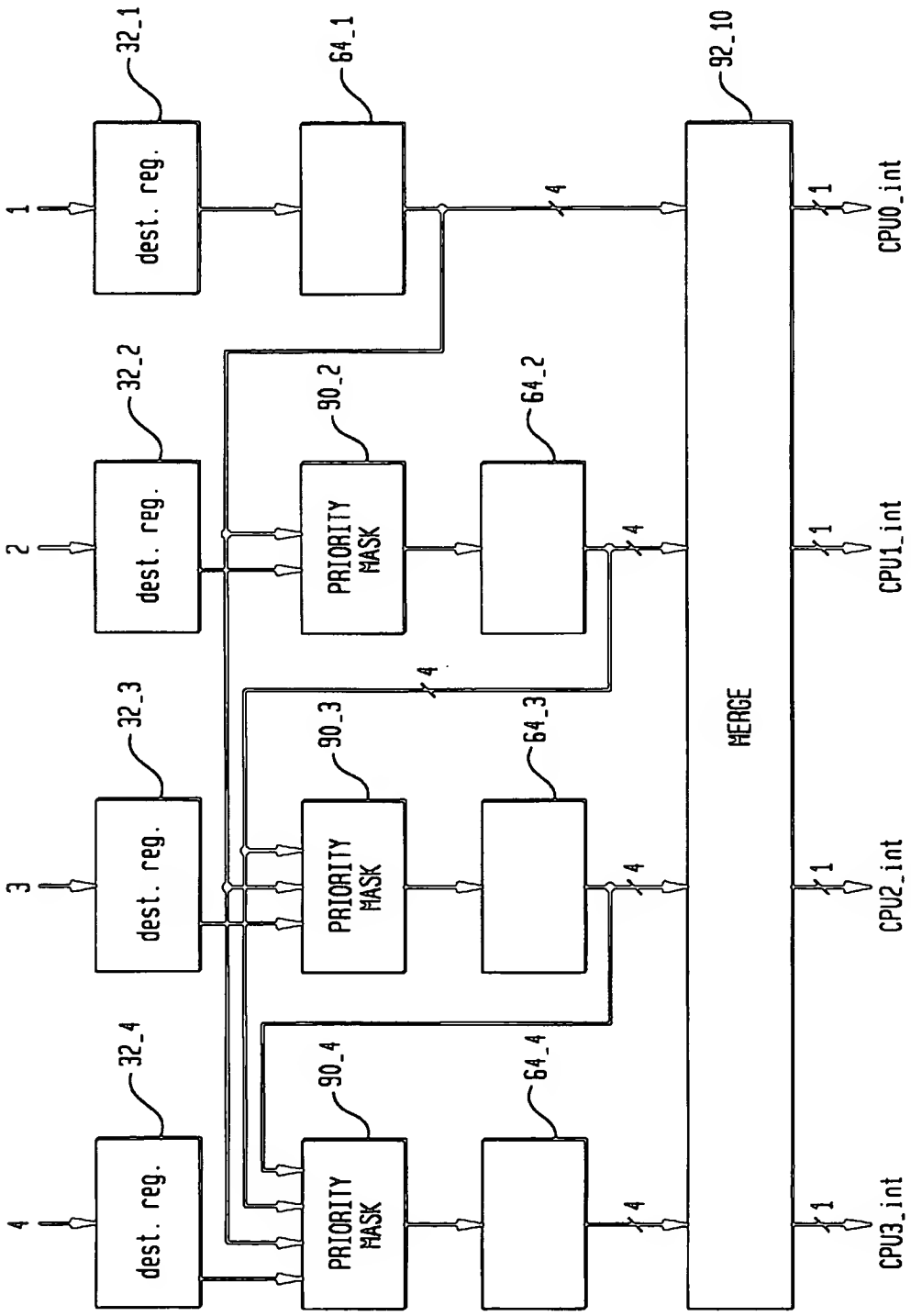

1
2
3
4
dest. reg.
dest. reg.
dest. reg.
dest. reg.
32_1
32_2
32_3
32_4
64_1
90_2
64_2
90_3
64_3
90_4
64_4
PRIORITY MASK
PRIORITY MASK
PRIORITY MASK
92_10
MERGE
4
4
4
4
4
4
4
4
1
1
1
1
CPU0_int
CPU1_int
CPU2_int
CPU3_int

FIG. 11

METHOD AND APPARATUS FOR DISPATCHING MULTIPLE INTERRUPT REQUESTS SIMULTANEOUSLY

RELATED APPLICATION

This invention is related to U.S. patent application Ser. No. 08/822,183, entitled "Method and Apparatus for Selecting a Nonblocked Interrupt Request," filed on even date herewith for Hong-Chich Chou, Jerng-Cherng Fan, Won-Yih LIN and Ching-Chin Huang. The contents of the above-noted application is incorporated herein by reference.

FIELD OF THE INVENTION

The present invention relates to interrupt processing in processor-based systems and devices. More particularly, the invention relates to interrupt processing which permits simultaneous delivery of multiple interrupts and/or permits selection of a non-blocked interrupt request in a multiprocessor system.

BACKGROUND OF THE INVENTION

FIG. **1** shows a conventional multiprocessor system **10** with distributed interrupt control. The system **10** includes four processors (CPUs) CPU1, CPU2, CPU3 and CPU4 each coupled to a CPU bus **12** and interrupt bus **16**. The system **10** further includes I/O controller **14** coupled to the interrupt bus **16**. The I/O controller **14** directs the transfer of data to and from peripheral devices such as displays, printers and disk drives. Each of the CPUs and the I/O controller **14** have an embedded Advanced Programmable Interrupt Controller (APIC) associated therewith coupled to the interrupt bus **14**. Each APIC includes a hardware state machine for processing interrupt requests in a conventional manner. For example, the APIC of I/O controller **14** broadcasts interrupts to all of the CPUs over the interrupt bus **16**. A problem with the distributed interrupt control of system **10** is that each of the CPUs and I/O controller must include a separate APIC, which unduly increases the cost and complexity of the system.

FIG. **2** shows a conventional multiprocessor system **20** with centralized interrupt control. The system **20** includes four processors (CPUs) CPU1, CPU2, CPU3 and CPU4 coupled to a CPU bus **22**, and a Multiple Processor Interrupt Controller (MPIC) **24** coupled to a Peripheral Component Interconnect (PCI) bus **26** and an interrupt bus **25**. The CPU bus **22** and PCI bus **26** are interconnected by a CPU-to-PCI bus bridge **28** which regulates the bidirectional flow of data between the PCI bus **26** and CPU bus **22**. The MPIC **24** dispatches a given interrupt to the appropriate destination CPU of the interrupt rather than broadcasting the interrupt to all CPUs. The MPIC is generally configured such that only the current highest priority interrupt is dispatched at any particular time. An exemplary MPIC is described in greater detail in Donald W. McCauley, "Power PC Multiprocessor Interrupt Controller (MPIC)," IBM Power Personal Systems, Austin Tex., pp. 1–22, Aug. 14, 1995, which is incorporated by reference herein.

The multiprocessor systems **10** and **20** of FIGS. **1** and **2** are referred to as symmetrical multiprocessor systems because each of the CPUs CPU1, CPU2, CPU3 and CPU4 has the ability to receive and process interrupt requests. For example, the MPIC **24** in system **20** receives all of the interrupt requests, and dispatches them to the appropriate CPUs such that the interrupt requests are processed evenly across the CPUs. The above-cited IBM reference includes a specification referred to as OpenPIC which facilitates this type of multiprocessor interrupt processing. In accordance with the OpenPIC specification, each interrupt request has a destination register, a vector register and a priority register associated therewith. The destination register is used to identify which CPUs can service a particular interrupt request, the vector register holds the starting address in system memory of the interrupt service routine for the particular interrupt request, and the priority register indicates the relative priority of the particular interrupt request. In operation, the MPIC **24** detects an interrupt signal from an I/O device coupled to the PCI bus **26**, and determines which CPU or CPUs to which the corresponding interrupt request should be dispatched using the information in the above-noted destination register. A CPU to which the interrupt is dispatched detects the interrupt request, reads the interrupt vector to determine the starting address of the interrupt service routine, and executes the interrupt service routine.

A number of significant problems limit the efficiency of conventional multiprocessor interrupt controllers such as MPIC **24** of FIG. **2**. For example, a conventional interrupt controller can usually select and dispatch only the current highest priority interrupt request at a given time. After the current highest priority interrupt request is dispatched, but before it is received and accepted by the destination CPU, the interrupt controller will generally prevent the selection and dispatch of any further interrupt requests. It is therefore usually not possible to dispatch multiple interrupt requests simultaneously to different destination CPUs. Moreover, the current highest priority interrupt may be blocked because all of its possible destination CPUs are busy either handling interrupt requests or performing other tasks having a higher priority than the current highest priority interrupt. Conventional interrupt controllers are unable to mask blocked interrupt requests that cannot be dispatched so as to avoid preventing the selection and dispatch of non-blocked interrupt requests. These problems substantially undermine the efficiency of conventional interrupt controllers and thereby degrade the performance of multiprocessor systems which include such interrupt controllers.

As is apparent from the above, there is a need for a multiprocessor interrupt controller in which multiple interrupt requests can be dispatched simultaneously to different destination CPUs, and in which blocked interrupt requests which cannot be dispatched at a particular time are masked to thereby allow the selection and dispatch of non-blocked interrupt requests.

SUMMARY OF THE INVENTION

The present invention provides an improved interrupt processing method and apparatus particularly well-suited for use in a multiprocessor interrupt controller. The interrupt controller may be configured to dispatch multiple interrupt requests simultaneously. The interrupt controller may also be configured to select a highest priority non-blocked interrupt request from multiple pending non-blocked interrupt requests, such that when all possible destination processors of the highest priority interrupt request are unavailable, the remaining non-blocked interrupt requests can be selected and dispatched.

One aspect of the invention relates to the simultaneous delivery of multiple interrupt requests in a multiple processor system. As noted above, conventional interrupt controllers generally dispatch only a single interrupt request at a time, and system performance is therefore limited. An

interrupt controller in accordance with the present invention may include a pending interrupt register which latches interrupt requests received from various external sources. The outputs of the pending interrupt register are applied to a priority comparc trec which includes multiple levels of comparators and serves to prioritize the received interrupt requests. An exemplary embodiment for use in a system with four processors identifies four higher priority interrupt requests, including the highest priority interrupt request. These requests are supplied to a destination selection circuit which utilizcs destination registers for storing a four-bit indicator for each of the four identified interrupt requests with each bit of the indicator specifying whether or not a particular one of the four processors is a possible destination register for the corresponding interrupt request. The destination processor information for thc first priority interrupt (which is also the higbest priority) is applied to a first interrupt dispatcher which determines the processor to which the first priority interrupt request will be dispatched. Similar determinations are made for the second, third and fourth identified interrupt requests, but with the destination register contents masked to prevent those processors already selected to receive an interrupt from being considered for duplicate selection. The destination selection circuit thus attempts to determine a unique destination processor for each of the four identified interrupt requests identified by the priority compare tree circuit. In this manner, the interrupt controller is able to ensure that there will be no conflict in the destination processors of the four identified interrupt requests, and these multiple interrupt requests can therefore be dispatched to different processors simultaneously. This provides considerable improvement in overall system efficiency relative to conventional interrupt controllers which are able to dispatch only a single interrupt request at a time.

Another aspect of the invention relates to an interrupt control technique in which blocked or otherwise non-dispatchable interrupt requests are masked and thereby prevented from being selected for dispatch. Instead of selecting the highest priority interrupt request from all pending interrupt requests as in a conventional interrupt controller, the interrupt control technique of the present invention involves selecting the highest priority interrupt requcst from all non-blocked interrupt requcsts. An cxemplary metbod includes the steps of masking all blocked interrupt requests that cannot be dispatched at a particular time because all destination processors associated with the blocked interrupt requests are processing other interrupts, performing other bigher priority tasks or are otherwise unavailable. The remaining non-blocked interrupt request having the highest priority is then selected for dispatch to an available processor. The masking process may utilize processor availability indicators supplied from the destination processors, as well as identifiers of the particular interrupt requests which have been accepted by each of the processors as a result of a previous interrupt selection process. Interrupts accepted in a previous sclection process may bc rcmovcd from a current interrupt selection process, such that performance-limiting repetition is considerably reduced.

These and other features and advantages of the present invention will become more apparent from the accompanying drawings and the following detailed description.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1** and **2** show conventional multiprocessor systems with distributed interrupt control and centralized interrupt control, respectively.

FIG. **3** shows a multiprocessor system with an interrupt controller in accordance with the present invention.

FIG. **4** illustrates an exemplary set of registers suitable for use in the interrupt controller of FIG. **3**.

FIGS. **5** and **6** show the exemplary interrupt controller of FIG. **3** in greater detail, which allows the sclection and dispatch of non-blocked interrupt rcquests.

FIG. **7** shows an exemplary interrupt masking and routing circuit suitable for use in the interrupt controller of FIG. **6**.

FIGS. **8**A, **8**B, **8**C, **8**D, and **8**E show exemplary destination mask, acceptance mask, task priority mask, priority compare tree and interrupt dispatcher circuits suitable for use in the interrupt masking and routing circuit of FIG. **7**.

FIG. **9** shows an exemplary embodiment of an interrupt controller with simultaneous delivery of multiple interrupts in accordance with the invention.

FIG. **10** shows an exemplary priority compare tree circuit suitable for use in the interrupt controller of FIG. **9**.

FIG. **11** shows an exemplary destination selection circuit suitable for use in thc interrupt controllcr of FIG. **9**.

DETAILED DESCRIPTION OF THE INVENTION

The present invention will be illustrated below using an exemplary CPU-based system. It should be understood, bowever, that the described techniques are more generally applicable to any other data processing application in which multiple interrupt requests are processed. Thc terms "CPU" and "processor" as used herein are intended to include any device which can accept an interrupt request and execute associated interrupt handling instructions. The term "interrupt request" is intended to include any physical signal asserted by an external device indicating tbat tbe device has reached a particular state and requires processing. The term "interrupt controller" refers to a device which coordinates interrupt requests and routes the requests to an appropriate destination CPU or CPUs.

FIG. **3** shows a multiprocessor system in which the present invention may be implemented, and illustrates the basic steps of interrupt processing. The system includes an interrupt controllcr **30** whicb is configurcd to detect and latch in sixtccn different interrupt requcsts int0 through int15. These interrupt requests may be received from peripheral devices such as a keyboard, mouse, monitor, printer and disk drive as well as other elements of a computer system. The interrupt controller **30** compares the relative priority and dispatchability of all pending interrupt requests in a manner to be described below, and dispatches a selected interrupt via interrupt signal lines int_cpu**0**, int_cpu**1**, int_cpu**2** or int_cpu**3** to an appropriate destination CPU of the group of CPUs including CPU**0**, CPU**1**, CPU**2** or CPU**3**. A given destination CPU acknowledges to the interrupt controller **30** that an interrupt vector dispatched thereto has been received. This acknowledgement process generally involves the destination CPU reading the interrupt vector of the dispatched interrupt from thc corresponding vcctor rcgistcr of the interrupt controller. The destination CPU sends an end-of-interrupt (EOI) indication to the interrupt controller to indicate that it has completed processing the interrupt service routine. It should be noted that the number of interrupt requests and CPUs is exemplary only, and that the interrupt processing techniques of the invention are applicable to a wide variety of alternative configurations.

FIG. **4** shows an exemplary set of internal registers of the interrupt controller **30** of FIG. **3**. Each of the interrupt requests int0 through int15 has a destination register **32**-*i*, a vector register **34**-*i* and a priority register **36**-*i* associated

therewith, where i=0, 1, 2 . . . 15. A given destination register **32-*i*** contains information identifying the particular CPU or CPUs to which the corresponding interrupt request may be directed. A given vector register **34-*i*** contains information identifying an interrupt vector for the corresponding interrupt request. The interrupt vector generally indicates a starting address in system memory for an interrupt service routine for the corresponding interrupt request. A given priority register **36-*i*** contains information identifying the priority of the corresponding interrupt request relative to the other interrupt requests. The interrupt controller **30** also includes a CPU task priority register **38-*j*** for each CPUj, where j=0, 1, 2, 3. A given CPU task priority register **38-*j*** includes information regarding a threshold priority level of interrupt acceptance for the corresponding CPUj. That is, a given CPU task priority register indicates the priority of the task currently being performed by the corresponding CPU, and therefore the minimum level of interrupt priority which the CPU will accept. The destination registers **32-*i***, vector registers **34-*i***, priority registers **36-*i*** and CPU task priority registers **38-*j*** may all be implemented as software programmable registers. These registers may be configured in a conventional manner such as that described in the above-cited IBM MPIC reference.

FIG. **5** shows the exemplary interrupt controller **30** of FIG. **3** in greater detail. The interrupt controller **30** includes an interrupt selection and routing process circuit **40** which receives as inputs the int0 through int15 interrupt requests as well as the contents of the destination registers **32**, vector registers **34**, priority registers **36** and task priority registers **38**. The process circuit **40** uses these inputs to generate an interrupt signal int_cpuj for each of the j CPUs. FIG. **6** shows a more detailed block diagram of the interrupt selection process circuit **40** within the exemplary interrupt controller **30**. The process circuit **40** includes an interrupt pending register **41** which receives the into through int15 interrupt requests as inputs. The register **41** latches these external interrupt requests periodically. The latched interrupt requests are delivered as an interrupt signal ipr[**15:0**] with each bit ipr[i] indicating the presence or absence of the ith interrupt request when the inputs int0 through int15 were last latched into register **41**. The ipr[**15:0**] signal is delivered to an interrupt masking and routing circuit **42** which also receives the contents of the destination registers **32**, vector registers **34**, priority registers **36** and CPU task priority registers **38**.

The interrupt masking and routing circuit **42** processes the ipr[**0**] through ipr[**15**] interrupts using the contents of the registers **32**, **34**, **36** and **38** to select the highest priority dispatchable or non-blocked interrupt for delivery to a particular CPU or CPUs. A selected interrupt is delivered to one of j interrupt request registers (IRRs) **44-*j***, j=0, 1, 2, 3. Each of the j IRRs is associated with one of the j CPUs. The interrupt masking and routing circuit **42** delivers the selected interrupt as a one-bit signal cpuj_valid_int, an interrupt identifier h_int_id, the interrupt vector h_int_vec and the interrupt priority h_int_pri. The signals h_int_id, h_int_vec and h_int_pri are supplied to each of the IRRs **44-*j***, and the signal cpuj_valid_int is used to latch these values for a selected interrupt into a particular one of the IRRs **44-*j*** corresponding to an appropriate destination CPU for that interrupt.

The process circuit **40** further includes j interrupt service registers (ISRs) **46-*j***, each of which receives an isrj load signal from the corresponding jth CPU. The ISRs **46-*j*** also receive interrupt priority signal irrj_int_pr, which is initially 0. The output of the jth ISR drives one input of a comparator **48-*j***, while interrupt priority signal irrj_int_pr from the jth IRR drives the other input of comparator **48-*j***. The output of comparator **48-*j*** is 1 if the irrj_int_pr input is greater than the input from the jth ISR which enables the corresponding jth CPU as interrupt signal int_cpuj. Each of the j IRRs also generates an int_id_accepted by_cpuj signal and a cpuj_busy bit which are both fed back to the interrupt masking and routing circuit **42**. These signals provide information regarding an interrupt already received by the corresponding CPU.

Whenever a new cpu_j_valid_int signal is produced by the interrupt masking and routing circuit **42** entering an IRR_j **44-*j***, the corresponding IRR_j will generate a cpu_j_busy_bit signal feedback to circuit **42**. When cpu_i acknowledges this interrupt request, the cpu_j_busy_bit signal is released, and IRR_j **44-*j*** generates an int_id_accept_by_cpu_j signal feedback to circuit **42**. Finally, when cpu_j finishes servicing the interrupt request (EOI), the signal int_id_accept_by_cpu_j will be released.

FIG. **7** shows a more detailed diagram of an exemplary interrupt masking and routing circuit **42** suitable for use in the interrupt selection and routing process circuit **40** of FIG. **6**. The interrupt masking and routing circuit **42** includes a destination mask circuit **52** which receives the [3:0] bits of each of the destination registers **32**, and a cpuj_busy bit from each of the four IRRs **44-*j***. FIG. **8A** shows the destination mask circuit **52** in greater detail. The destination mask circuit **52** includes multiple AND gates **53-*i*** for masking the [3:0] destination register bits of those interrupt requests which have a destination CPU which is busy as indicated by the four cpuj_busy bits. The outputs of the multiple AND gates **53-*i*** correspond to four-bit available_inti_destination signals for each of the interrupt requests. A given ith four-bit available_inti_destination signal, which is used in task priority mask **56**, will indicate those of all possible CPU destinations for the ith interrupt which are not busy. This masking ensures that the ith interrupt will not be directed to a busy CPU.

The interrupt masking and routing circuit **42** of FIG. **7** further includes an acceptance mask circuit **54** which receives as inputs the four four-bit int_id_accepted_by_cpuj signals supplied from the four IRRs **44-*j***. FIG. **8B** shows the acceptance mask circuit **54** in greater detail. The acceptance mask circuit **54** includes j decoders **50-*j***, i OR gates **51-*i***, and i NOT & AND gates **55-*i*** configured as shown. Each of the four-bit int_id_accepted_by_cpuj is decoded into i independent signals by each of the decoder **50-*j***. Each of the i OR gates receives one bit signal from each of the decoder **50-*j***. The i OR gates generate i inti_accepted signals which are supplied to i NOT gates and to i AND gates (**55-*i***) together with inti pending signals.

These circuits efficiently mask the interrupts already accepted by the four CPUs and generate new masked inti pending signals that will be used in compare tree circuit **60**. FIG. **8C** shows the task priority mask circuit **56** of FIG. **7** in greater detail. The circuit **56** includes i circuits, each with four 2-input compare circuits **57-*i*** and four 2-input AND gates **59-*i***.

Each of the 2-input compare circuits **57-*i*** receives as inputs an inti priority and a cpuj task priority. Circuit **57-*i*** outputs a 0 if the cpuj task priority is higher than the inti priority. Otherwise, circuit **57-*i*** outputs a 1. Each of the four 2-input AND gates **59-*i*** receive as an input the corresponding one bit of the four-bit available_inti_destination signal of each interrupt i from the destination mask circuit **52** and also receives the output from circuit **57-*i***. The AND **59-*i***

gates generate masked inti destination signals, which indicate whether one of the possible destination CPUs for the ith interrupt is free, and is able to service interrupts.

The masked inti destination signals from the task priority mask circuit **56** and the interrupt priority registers **36** (inti id information) are applied to the priority compare tree **60**, configured as shown in FIG. **8D**. The compare tree circuit **60** includes 16 latches **60-0** to **60-15**, and uses pending interrupt signals that are generated from circuit **54** to decide which interrupt i priority can join the comparison. Circuit **60** also includes multiple layer comparator **60-16** to **60-30**, illustratively, which determines the highest priority pending interrupt.

The interrupt dispatcher **64** is used to choose one of the valid destination cpuj to dispatch the non-blocked highest priority pending interrupt. The interrupt vector information is also passed down through circuit **64**. As shown in FIG. **8E**, when signal **64-1** is 1, the output from gates **64-*i*** will be 0. Similarly, when **64-1** is 0 and **64-2** is 1, the gates **64-6** and **64-7** will output 0, and so choose which of the cpui is valid and ready to service the pending interrupt.

The above-described exemplary interrupt controller provides improved performance in the following manner. The interrupt masking and routing circuit **42** selects and dispatches a given interrupt request to a particular IRR **44-*j*** associated with the jth destination CPU. This interrupt is compared with the current in-service interrupt of the destination CPU as stored in the corresponding ISR **46-*j***. If the priority of the interrupt in IRR **44-*j*** is greater than the priority of the current in-service interrupt stored in ISR **46-*j***, then the comparator **48-*j*** enables the interrupt to the jth destination CPU. Otherwise, the interrupt in IRR **44-*j*** is not sent out to the destination CPU until that CPU completes its current in-service interrupt processing operation. In addition, once the interrupt masking and routing circuit **42** selects and dispatches a given interrupt request to a particular IRR **44-*j***, this interrupt request will not be used in the next selection of interrupts regardless of whether or not it is accepted by its destination CPU. This feature is provided in the controller **30** via the int_id_accepted_by_cpuj signal and cpuj_busy bit applied from the IRR **44-*j*** to the acceptance mask circuit **54** and destination mask circuit **52**, respectively. A current dispatched interrupt request in the interrupt controller **30** of the present invention will therefore not block the dispatch of other interrupt requests to other destination CPUs, thereby increasing interrupt processing speed and overall system efficiency.

The conditions for the dispatch of a given interrupt request in the interrupt controller **30** of FIG. **6** may be summarized as follows: (1) at least one of the j IRRs **44-*j*** corresponding to the jth CPU is empty; (2) an interrupt request already dispatched to a given destination CPU will not be selected in the next interrupt selection process; and (3) out of all dispatchable or non-blocked interrupt requests, the highest priority request is selected for dispatch. Conditions (1) and (2) ensure that all of the interrupt requests that are selected by interrupt controller **30** will be dispatched to their destination CPUs regardless of whether or not other higher priority interrupt requests have been blocked. The valid new inti pending and destinations signals applied to the priority compare tree **60** and interrupt dispatcher circuit **64** thus indicate whether at least one of the possible destination CPUs of the ith interrupt is available and whether the ith interrupt has been dispatched as a result of a previous selection process. The interrupt controller is therefore configured to select a dispatchable interrupt request from multiple pending interrupt requests.

An interrupt controller in accordance with the present invention provides a number of advantages relative to prior art controllers. As noted above, prior art controllers are configured such that when all possible destination CPUs of the highest priority interrupt requests are busy, the highest priority interrupt request cannot be dispatched and it blocks other interrupt requests from being dispatched. Prior art interrupt controllers also block other interrupt requests from being selected and dispatched when the highest priority interrupt request is issued but not yet received by the destination CPU. The present invention overcomes these problems and allows non-blocked interrupt requests to be dispatched even if the highest priority interrupt request is blocked or not yet received or accepted by the destination CPU.

FIG. **9** shows another exemplary interrupt controller **70** used to deliver interrupts selected from int0 through int15 to the appropriate destination CPUs CPU0, CPU1, CPU2 and CPU3 as in the previous embodiment. The interrupt controller **70** as illustrated includes the interrupt 0–15 destination registers **32**, the interrupt 0–15 priority registers **36**, and the pending interrupt request register **41**, all of which operate in the manner previously described. The controller **70** may also include other elements of the controller of FIGS. **4–7**, such as the vector registers **34**. These other elements are omitted from FIG. **9** for clarity of illustration. The interrupt controller **70** further includes a priority compare tree circuit **72** and a destination selection circuit **74**. The priority compare tree circuit **72** determines the relative priority of the received interrupt requests, and the destination selection circuit **74** determines a unique destination CPU for each of a number of selected interrupts including the highest priority interrupt. The circuits **72**, **74** permit the interrupt controller **70** to select and simultaneously dispatch not only the highest priority interrupt request but also other lower-priority interrupt requests latched into the interrupt pending register **41**.

FIG. **10** illustrates the priority compare tree circuit **72** of interrupt controller **70** in greater detail. The pending int bit from pending interrupt register **41** (FIG. **9**) loads the corresponding sixteen registers INT0 through INT15 via circuits **79-*i***, for i=0 to 15. Each register (INT0–INT15) latches a corresponding one of the received int0 through int15 interrupt requests. The outputs of the registers INT0 through INT15 are applied to a first level of comparators **80-*k*** and **81-*k***, k=1, 2, 3, 4. Each of the first level comparators **80-*k*** or **81-*k*** perform a pair-wise comparison of the priorities of interrupt requests int0 through int15. The outputs of the first level comparators **80-*k*** and **81-*k*** are applied to a second level of comparators **82-*k*** which perform a pair-wise comparison of the results from the first level of comparators. The outputs of the comparators **82-*k*** are routed to a third level of comparators **84-*k*** in the manner shown in FIG. **10**, with the solid line showing the routing for one comparison result and the dashed line showing the routing for the opposite result. The results from the third level of comparators **84-*k*** are similarly routed to a fourth level of comparators **86-*k*** in accordance with the results of the third level comparisons. The result of the multiple-level comparison process is that the output of comparator **86-1** represents the highest priority interrupt request, and the outputs of comparators **86-2**, **86-3** and **86-4** represent other three selected interrupts. These interrupt requests are supplied to the four inputs of the destination selection circuit **74**. If additional layers of comparators are provided, the interrupts outputted from comparators **86-2**, **86-3** and **86-4** would be the second, third and fourth highest priority interrupts, respectively.

FIG. 11 shows the destination selection circuit 74 in greater detail. The four selected interrupt requests are applied to corresponding destination registers 32-*i* which will store information regarding the possible destination CPUs of each of the interrupt requests. For example, the destination register may be used to store a four-bit indicator, with each bit of the indicator specifying whether or not a particular one of the four CPUs is a possible destination register for the corresponding interrupt request. Then, the destination processor information is applied to a first interrupt dispatcher 64-1, which determines the CPU to which the first priority interrupt request will be dispatched.

The destination selection circuit 74 also uses priority mask circuits 90-*i*, for i=2 to 4, to control for the fact that there may be overlap in the possible destination CPUs of each of the four selected priority interrupt requests. The destination CPU selected for the first priority interrupt request by the first interrupt dispatcher 64-1 is therefore supplied to a priority mask circuit 90-2 which prevents that CPU from being considered a possible destination CPU for the second selected priority interrupt request. A second interrupt dispatcher 64-2 uses the priority masked destination indication for the second selected priority interrupt request to determine an appropriate destination CPU for the second selected priority interrupt request. The outputs of dispatcher 64-1 and 64-2 are also supplied to a mask circuit 90-3 so that the destination CPUs determined for the first and second priority interrupt requests will not be considered for the third priority interrupt request. A third interrupt dispatcher 64-3 uses the priority and the contents of the third destination register 32-3 to determine an appropriate destination CPU for the third priority interrupt request. The outputs of dispatchers 64-1, 64-2, and 64-3 are also supplied to a mask circuit 90-4 so that the destination CPUs determined for the first, second and third priority interrupt requests will not be considered for the fourth priority interrupt request. A fourth interrupt dispatcher 64-4 uses the priority and the contents of the fourth destination register 32-4 to determine an appropriate destination CPU for the fourth priority interrupt request. The destination selection circuit thus attempts to determine a unique destination CPU for each of the four selected priority interrupt requests identified by the priority compare tree circuit 72. The four bit output signals from dispatcher 64-*i* (each jth bit corresponding to the jth CPU) will be sent into the corresponding jth OR gate (not shown) in a merge circuit 92_10, to generate the final cpuj_int signals.

In this manner, the interrupt controller 70 is able to ensure that there will be no conflict in the destination CPUs of the four selected priority interrupt requests, and these multiple interrupt requests can therefore be dispatched to different CPUs simultaneously. This provides considerable improvement in overall system efficiency relative to conventional interrupt controllers which are able to dispatch only a single interrupt request at a time. For example, assume that four interrupts are pending in the interrupt controller, and it takes four clock cycles to dispatch a single interrupt. The above-described multiple interrupt dispatching can be performed in about six clock cycles, resulting in a savings of 4×4−6 or 10 clock cycles.

In another embodiment, the present invention provides for dispatching multiple interrupt requests simultaneously, while also enabling non-blocked interrupt requests having a lower priority to be processed. This is achieved by replacing the priority compare tree 60 and dispatcher circuit 64 of FIG. 7 with the priority compare tree circuit 72 and destination selection circuit 74 of FIG. 9.

It should be understood that the foregoing description is merely illustrative of the invention. Numerous alternative embodiments within the scope of the appended claims will be apparent to those of ordinary skill in the art.

The claimed invention is:

1. A method of processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the method comprising the steps of:

prioritizing at least a portion of the interrupt requests to thereby identify a first priority interrupt request and a second interrupt request;

determining an appropriate destination processor for the first priority interrupt request; and

using the result of the determining step to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched approximately simultaneously to different destination processors, the step of using the result further including the step of applying the output of an interrupt dispatcher used to determine the destination processor for the first priority interrupt request to a mask circuit which masks a portion of the output of a destination register associated with the first priority interrupt request.

2. The method of claim 1 wherein the first priority interrupt request has the highest priority of the identified interrupt requests.

3. A method of processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the method comprising the steps of:

prioritizing at least a portion of the interrupt requests to thereby identify a first priority interrupt request and a second interrupt request;

determining an appropriate destination processor for the first priority interrupt request;

using the result of the determining step to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched approximately simultaneously to different destination processors;

using identifiers of the destination processors selected for the first and second interrupt requests in determining a destination processor for at least one additional interrupt request such that a destination processor other than those selected for the first and second interrupt requests may be selected for the at least one additional interrupt request, and the first, second and other interrupt requests can be dispatched approximately simultaneously to the different selected destination processors; and

using the identifiers to mask a portion of the output of a destination register associated with the at least one additional interrupt request.

4. The method of claim 3 wherein the step of using identifiers of the destination processors selected for the first and second interrupt requests in determining a destination processor for at least one additional interrupt request further includes using the identifiers to mask a portion of the output

of a destination register associated with the at least one additional interrupt request.

5. An interrupt controller for processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the interrupt controller comprising:

a priority compare tree circuit for prioritizing at least a portion of the interrupt requests to thereby identify a first priority interrupt request and a second interrupt request;

a destination selection circuit coupled to outputs of the priority compare tree circuit and operative to determine an appropriate destination processor for the first priority interrupt request, and to use the result of the determination to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched approximately simultaneously to different destination processors, wherein the destination selection circuit is further operative to apply the output of a priority decoder used to determine the destination processor for the first priority interrupt request to a mask circuit which masks a portion of the output of a destination register associated with the first priority interrupt request.

6. The interrupt controller of claim 5 wherein the first priority interrupt request has the highest priority of the identified interrupt requests.

7. The interrupt controller of claim 5 wherein the destination selection circuit is further operative to use identifiers of the destination processors selected for the first and second interrupt requests in determining a destination processor for at least one additional interrupt request such that a destination processor other than those selected for the first and second interrupt requests may be selected for the at least one additional interrupt request, and the first, second and additional interrupt requests can be dispatched approximately simultaneously to the different selected destination processors.

8. An interrupt controller for processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the interrupt controller comprising:

a priority compare tree circuit for prioritizing at least a portion of the interrupt requests to thereby identify a first priority interrupt request and a second interrupt request; and

a destination selection circuit coupled to outputs of the priority compare tree circuit and operative to determine an appropriate destination processor for the first priority interrupt request, and to use the result of the determination to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched approximately simultaneously to different destination processors,

wherein the destination selection circuit is further operative to use identifiers of the destination processors selected for the first and second interrupt requests in determining a destination processor for at least one additional interrupt request such that a destination processor other than those selected for the first and second interrupt requests may be selected for the at least one additional interrupt request, and the first, second and additional interrupt requests can be dispatched approximately simultaneously to the different selected destination processors, and

wherein the destination selection circuit is further operative to use the identifiers to mask a portion of the output of a destination register associated with the at least one lower priority interrupt request.

9. A method of processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the method comprising the steps of:

masking each interrupt request of said plurality of interrupt requests destined to processors that are unavailable at a given time;

prioritizing at least a portion of the remaining interrupt requests not masked in said step of masking to thereby identify a first priority interrupt request and a second interrupt request;

determining an appropriate destination processor for the first priority interrupt request; and

using the result of the determining step to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched contemporaneously to different destination processors, the step of using the result further including the step of applying the output of an interrupt dispatcher used to determine the destination processor for the first priority interrupt request to a mask circuit which masks a portion of the output of a destination register associated with the first priority interrupt request.

10. An interrupt controller for processing interrupt requests in a system having a plurality of processors, each of the interrupt requests having at least one destination processor associated therewith for servicing the interrupt request, the interrupt controller comprising:

a mask circuit for masking those of the plurality of interrupt requests for which the destination processors associated therewith are unavailable at a given time;

a priority comparison circuit for selecting from the remainder of the plurality of interrupt requests a particular interrupt to be delivered to an available destination processor associated with the particular interrupt request, and

a destination selection circuit coupled to the priority comparison circuit and operative to determine an appropriate destination processor for the first priority interrupt request, and to use the result of the determination to mask the second interrupt request such that a destination processor other than that selected for the first priority interrupt request may be selected for the second interrupt request, and the first and second interrupt requests can be dispatched contemporaneously to different destination processors, wherein the destination selection circuit is further operative to apply the output of a priority decoder used to determine the destination processor for the first priority interrupt request to a mask circuit which masks a portion of the output of a destination register associated with the first priority interrupt request.

* * * * *

US 20010032286A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2001/0032286 A1

Pawlowski (43) Pub. Date: Oct. 18, 2001

(54) MECHANISMS FOR CONVERTING INTERRUPT REQUEST SIGNALS ON ADDRESS AND DATA LINES TO INTERRUPT MESSAGE SIGNALS

(76) Inventor: Stephen S. Pawlowski, Beaverton, OR (US)

Correspondence Address:
Robert A Diehl
c/o Blakely Sokoloff Taylor Zafman LLP
12400 Wilshire Boulevard
7th Floor
Los Angeles, CA 90025 (US)

(*) Notice: This is a publication of a continued prosecution application (CPA) filed under 37 CFR 1.53(d).

(21) Appl. No.: 09/552,162

(22) Filed: Apr. 18, 2000

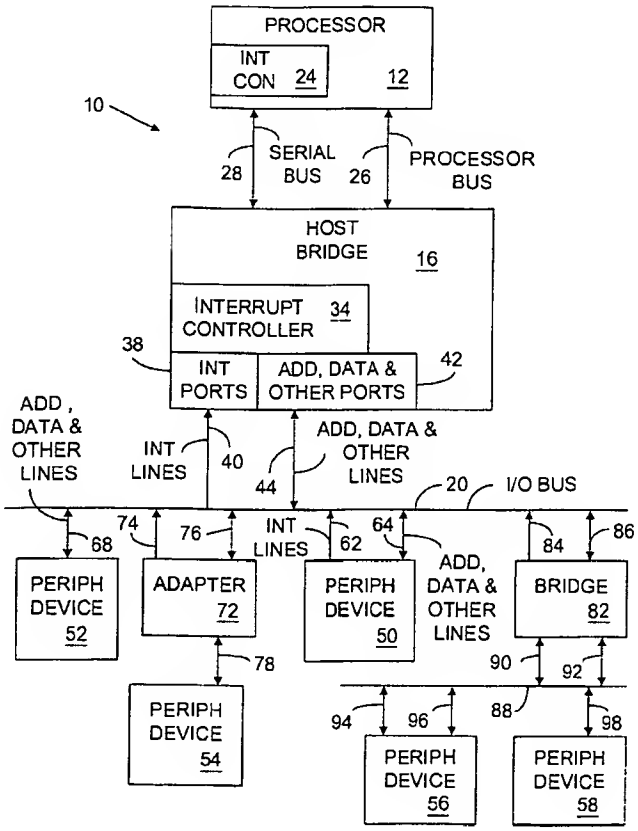
Publication Classification

(51) Int. Cl.[7] G06F 13/24

(52) U.S. Cl. 710/260

(57) ABSTRACT

In one embodiment of the invention, an apparatus includes address and data ports to receive an interrupt request signal in the form of address signals and data signals. The apparatus also includes decode logic to receive at least some of the address signals and data signals and provide a decoded signal at one of several decode output lines of the decode logic. A redirection table includes a send pending bit that is set responsive to the decode signal. In another embodiment, an apparatus includes dedicated interrupt ports to receive an interrupt request signal. The apparatus also includes address and data ports capable of receiving an interrupt request signal in the form of address signals and data signals, and decode logic to provide a decode signal at one of several decode output lines in response to reception of the interrupt request signal in the form of address signals and data signals. A redirection table includes a send pending bit to be set in response to either the interrupt request signal at the dedicated interrupt ports or in response to the decode signal.



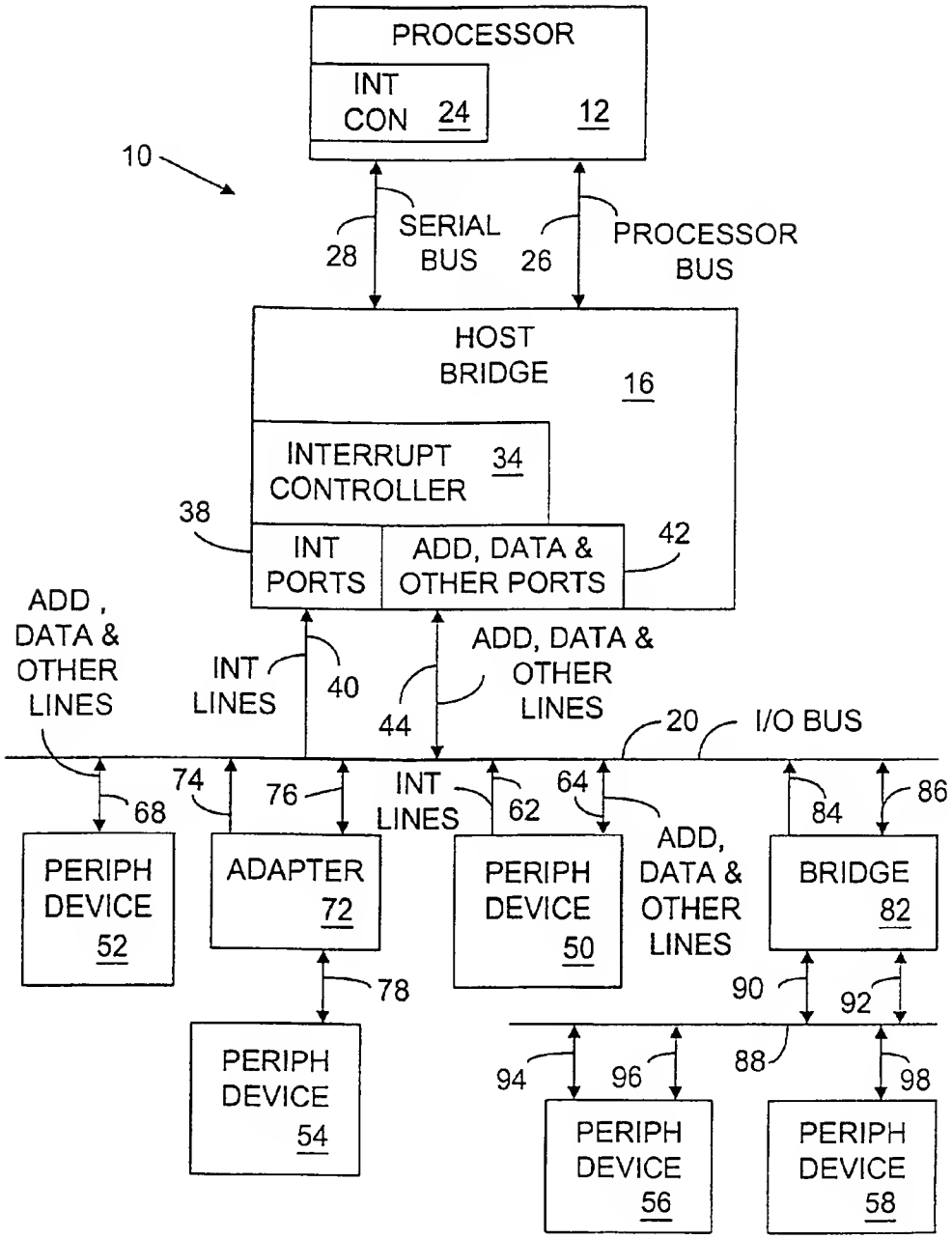

PROCESSOR
INT CON 24
12
10
28
SERIAL BUS
26
PROCESSOR BUS
HOST BRIDGE 16
INTERRUPT CONTROLLER 34
38
INT PORTS
ADD, DATA & OTHER PORTS
42
ADD , DATA & OTHER LINES
INT LINES
40
44
ADD, DATA & OTHER LINES
20
I/O BUS
68
74
76
INT LINES
62
64
84
86
PERIPH DEVICE 52
ADAPTER 72
PERIPH DEVICE 50
ADD, DATA & OTHER LINES
BRIDGE 82
78
90
92
PERIPH DEVICE 54
94
96
88
98
PERIPH DEVICE 56
PERIPH DEVICE 58

FIG. 1

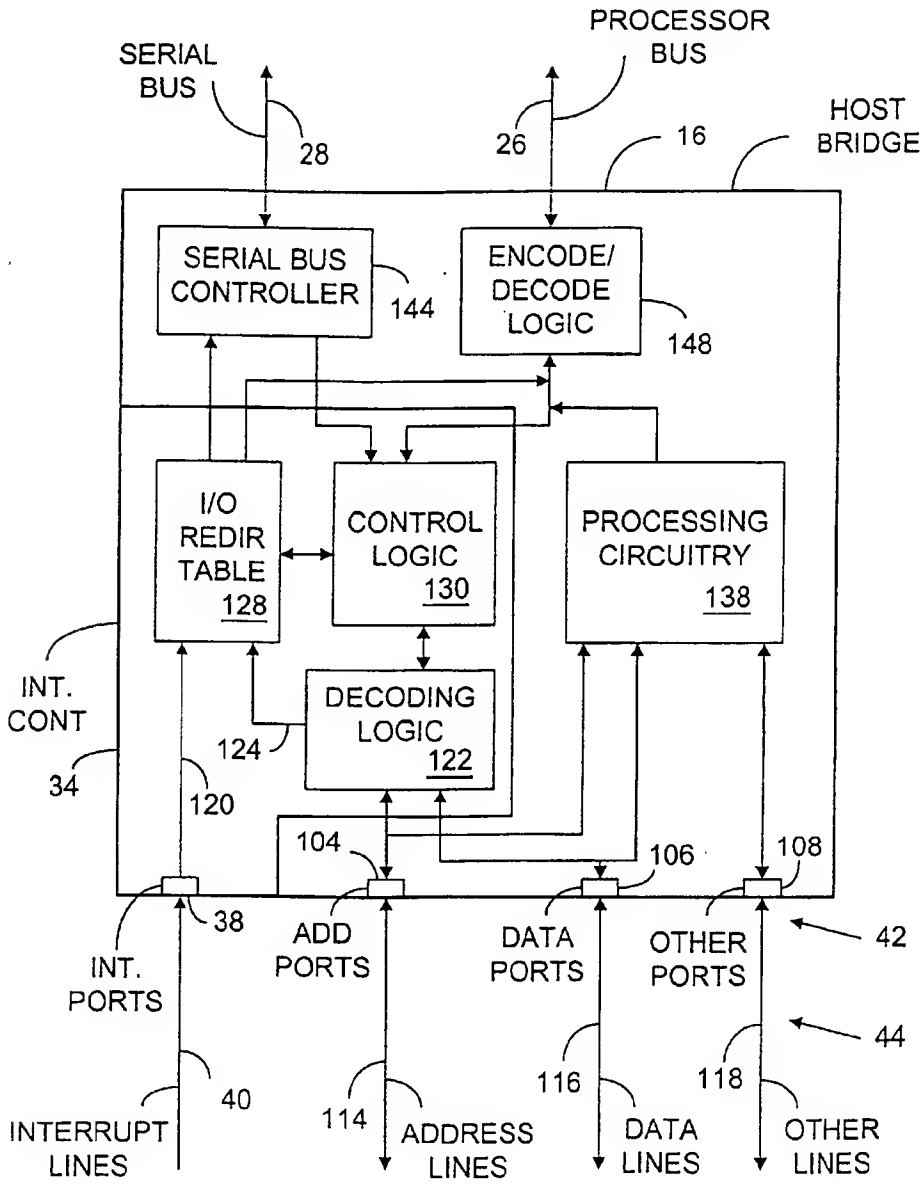

SERIAL BUS
28
PROCESSOR BUS
26
16
HOST BRIDGE
SERIAL BUS CONTROLLER
144
ENCODE/ DECODE LOGIC
148
I/O REDIR TABLE 128
CONTROL LOGIC 130
PROCESSING CIRCUITRY 138
INT. CONT
34
DECODING LOGIC 122
124
120
104
106
108
38
INT. PORTS
ADD PORTS
DATA PORTS
OTHER PORTS
42
44
40
114
116
118
INTERRUPT LINES
ADDRESS LINES
DATA LINES
OTHER LINES

FIG. 2

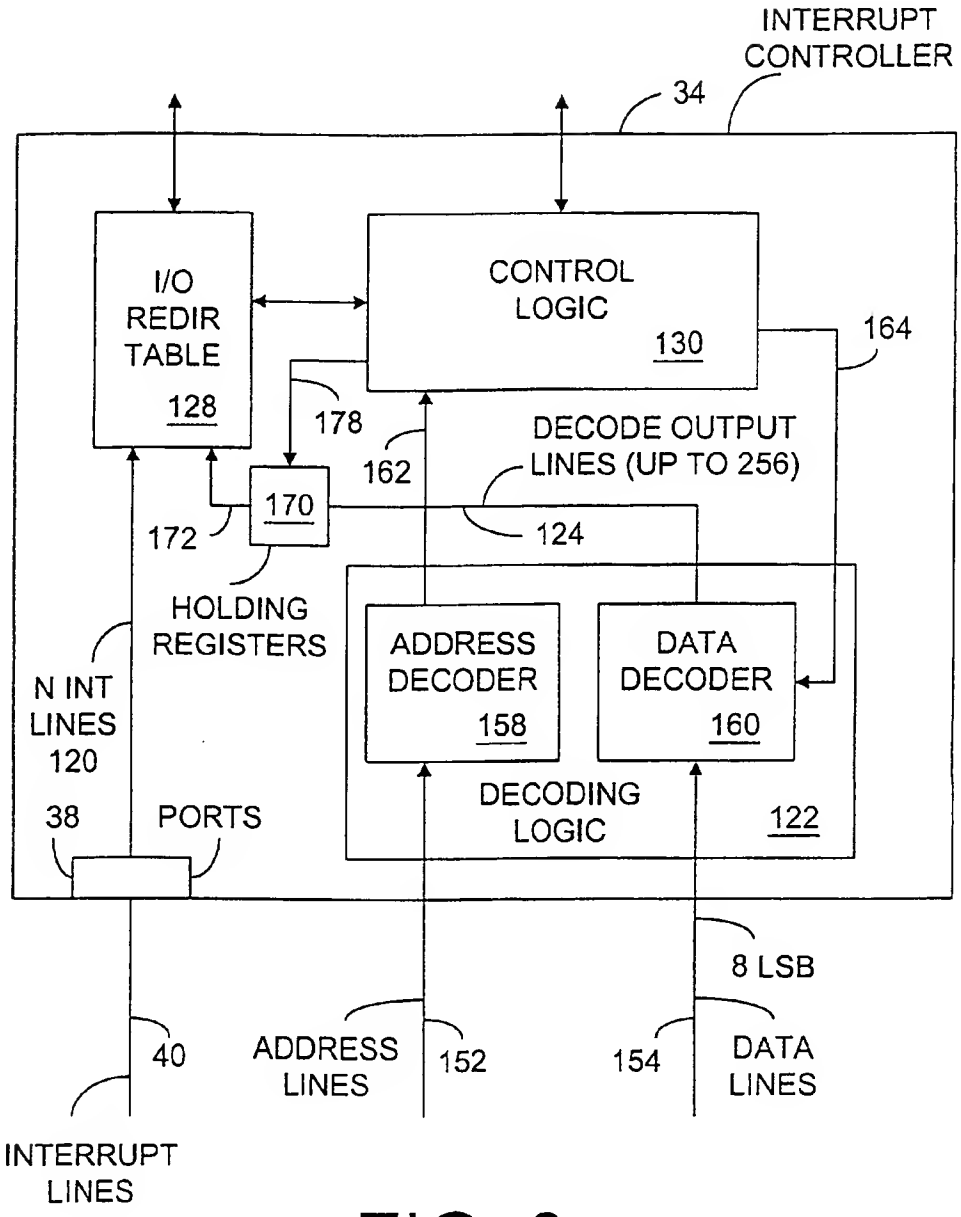

INTERRUPT CONTROLLER
34
I/O REDIR TABLE
128
CONTROL LOGIC
130
164
178
162
DECODE OUTPUT LINES (UP TO 256)
170
172
124
HOLDING REGISTERS
N INT LINES 120
ADDRESS DECODER
158
DATA DECODER
160
DECODING LOGIC
122
38
PORTS
8 LSB
40
ADDRESS LINES
152
154
DATA LINES
INTERRUPT LINES

FIG. 3

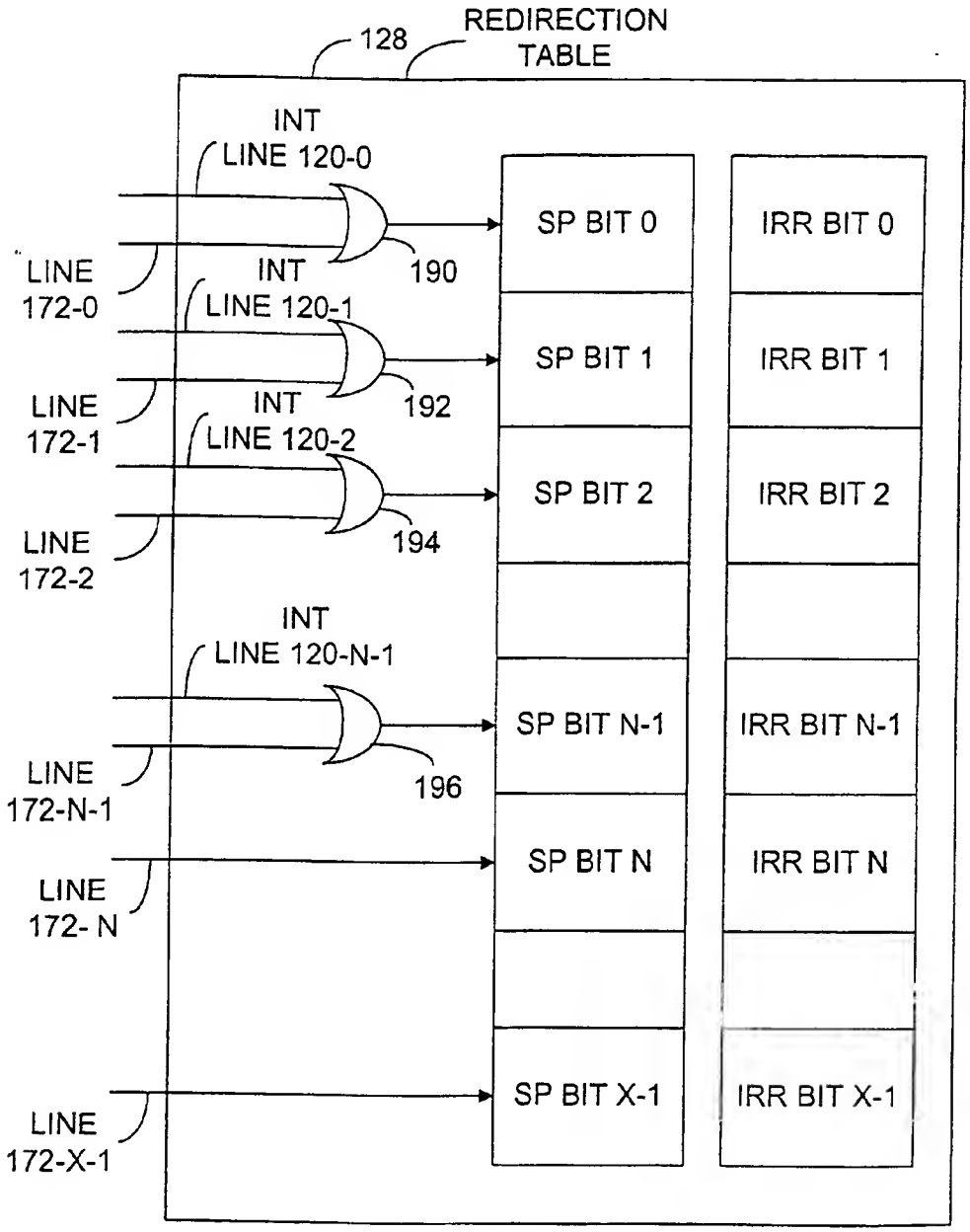

128
REDIRECTION TABLE
INT LINE 120-0
190
INT LINE 120-1
192
INT LINE 120-2
194
INT LINE 120-N-1
196
LINE 172-0
LINE 172-1
LINE 172-2
LINE 172-N-1
LINE 172- N
LINE 172-X-1
SP BIT 0
SP BIT 1
SP BIT 2
SP BIT N-1
SP BIT N
SP BIT X-1
IRR BIT 0
IRR BIT 1
IRR BIT 2
IRR BIT N-1
IRR BIT N
IRR BIT X-1

FIG. 4

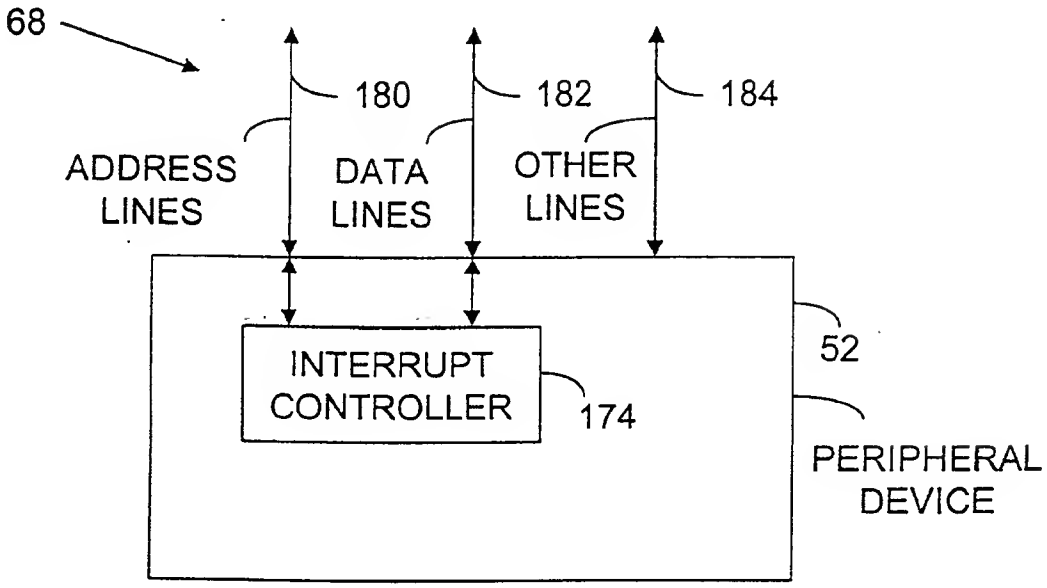

68
180
182
184
ADDRESS LINES
DATA LINES
OTHER LINES
INTERRUPT CONTROLLER
174
52
PERIPHERAL DEVICE

FIG. 5

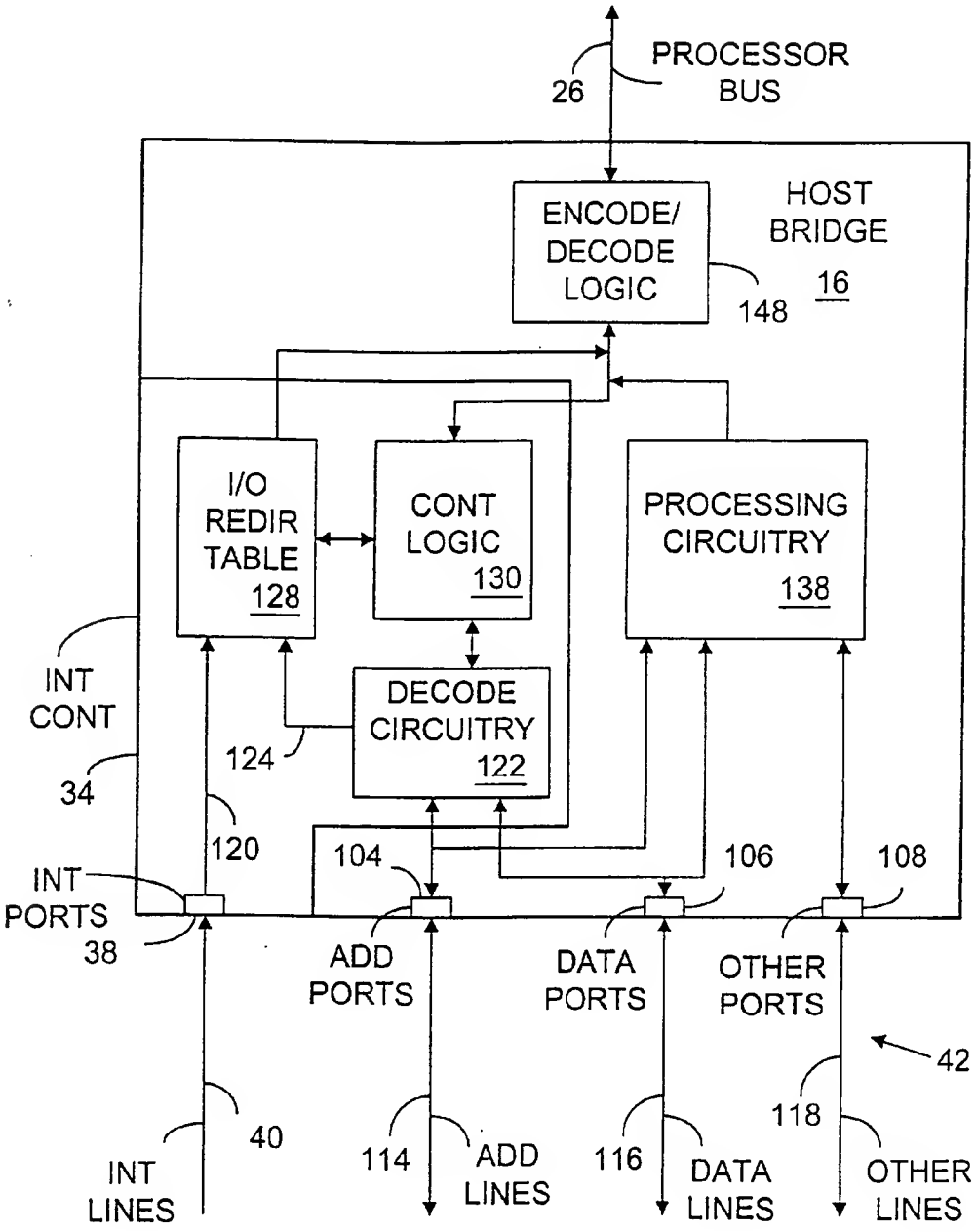

26
PROCESSOR BUS
ENCODE/ DECODE LOGIC
148
HOST BRIDGE
16
I/O REDIR TABLE
128
CONT LOGIC
130
PROCESSING CIRCUITRY
138
DECODE CIRCUITRY
122
124
INT CONT
34
120
104
106
108
INT PORTS
38
ADD PORTS
DATA PORTS
OTHER PORTS
42
40
118
114
ADD LINES
116
DATA LINES
INT LINES
OTHER LINES

FIG. 6

MECHANISMS FOR CONVERTING INTERRUPT REQUEST SIGNALS ON ADDRESS AND DATA LINES TO INTERRUPT MESSAGE SIGNALS

BACKGROUND OF THE INVENTION

[0001] 1. Technical Field of the Invention

[0002] The present invention relates to interrupts in a computer system.

[0003] 2. Background Art

[0004] A peripheral component interconnect (PCI) Local Bus Specification (Revision 2.1) ("PCI bus specification") has been developed to define a PCI bus. The PCI bus specification defines an interconnect mechanism and transfer protocol for devices on the bus. Additions or changes to the PCI specification are occasionally made. However, a guiding principle of the PCI specification is that of backward compatibility, wherein newer PCI systems will support older PCI peripheral devices.

[0005] Various devices including input and/or output (I/O) peripheral devices may seek to interrupt a processor in a computer system. When associated with a PCI bus, the devices are sometimes referred to as PCI agents. To interrupt a processor, the PCI agent may send one or more of interrupt request signals INTA#, INTB#, INTC#, or INTD# to an interrupt controller. The interrupt controller responds by providing an interrupt message to a processor. The interrupt controller receives the interrupt request signal through interrupt input pins. The interrupt input pins are sometimes called interrupt request (IRQ) pins, which are connected through IRQ lines to the PCI bus. There may be an interrupt router between the peripherals and the interrupt controller.

[0006] There are two types of signaling semantics for interrupt signals received by interrupt controllers: (1) edge triggered interrupt semantics and (2) level triggered interrupt semantics. With edge triggered interrupts, every time an edge (e.g., positive going edge) is detected at an interrupt input pin, the interrupt controller triggers an interrupt event. A problem with edge triggered interrupts is that the interrupt controller may miss an edge of a second interrupt if it occurs before a first interrupt is serviced. Accordingly, in the case of edge triggered interrupts, typically only one peripheral device is connected to the interrupt input pin.

[0007] With level triggered interrupts, a particular logical voltage level (e.g., a logical high voltage) at the interrupt input pin causes the interrupt controller to trigger an interrupt event. In the case of level triggered interrupts, more than one peripheral device may provide interrupt request signals to an input pin. However, the voltage level at the interrupt input pin provided by multiple peripheral devices is not different than the voltage level that is provided by only one peripheral device.

[0008] Accordingly, the interrupt controller cannot determine how many peripheral devices are providing an interrupt request signal merely by sensing the voltage level at the interrupt input pin. In response to detecting a change to the particular voltage level at the interrupt input pin, an interrupt message is sent to a processor and a state bit is set in an I/O redirection table in the interrupt controller. The state bit is reset when an end-of-interrupt (EOI) signal is received by the interrupt controller. If an interrupt signal having the particular voltage level is still detected at the interrupt input port after the EOI is received, another interrupt message is sent to a processor.

[0009] Interrupt controllers have a limited number of interrupt input pins. Under the present technology, as more peripheral devices are added to a computer system, the number of interrupt input pins will need to be increased or peripheral devices may need to wait longer for service of interrupts.

[0010] Accordingly, there is a need for an improved system for providing interrupt requests from peripheral devices to processors.

SUMMARY OF THE INVENTION

[0011] In one embodiment of the invention, an apparatus includes address and data ports to receive an interrupt request signal in the form of address signals and data signals. The apparatus also includes decode logic to receive at least some of the address signals and data signals and provide a decoded signal at one of several decode output lines of the decode logic. A redirection table includes a send pending bit that is set responsive to the decode signal.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] The invention will be understood more fully from the detailed description given below and from the accompanying drawings of embodiments of the invention which, however, should not be taken to limit the invention to the specific embodiments described, but are for explanation and understanding only.

[0013] **FIG. 1** is a block diagram representation of a computer system including a host bridge according to one embodiment of the present invention.

[0014] **FIG. 2** is a block diagram representation of one embodiment of the host bridge in the system of **FIG. 1**.

[0015] **FIG. 3** is a block diagram representation of one embodiment of the interrupt controller in the system of **FIG. 1**.

[0016] **FIG. 4** is a block diagram representation of one embodiment of send pending bits and related circuitry in the I/O redirection table of **FIGS. 2 and 3**.

[0017] **FIG. 5** is a block diagram representation of an exemplary peripheral device.

[0018] **FIG. 6** is a block diagram representation of an alternative embodiment of a host bridge in the system of **FIG. 1**.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

[0019] Referring to **FIG. 1**, a computer system **10** includes a processor **12**, a host bridge **16**, and an I/O bus **20**, which may be implemented according to PCI specifications. Processor **12** includes an interrupt controller **24**, which may be an advance programmable interrupt controller (APIC). Processor **12** is coupled to host bridge **16** through a processor bus **26** and a serial bus **28**, which may be an APIC serial bus. In some embodiments, host bridge **16** is referred to as a North Bridge and processor bus **26** is referred to as a front side bus or parallel bus.

[0020] Serial bus **28** may provide interrupt messages from an interrupt controller **34** in host bridge **16** to interrupt controller **24**. Interrupt controller **34** may be an APIC. Serial bus **28**, which may include two data conductors and a clock signal conductor, may also provide signals from processor **12** to interrupt controller **34**, such as end-of-interrupt (EOI) signals. In multi-processor systems, serial bus **28** may also be used in lowest priority interrupt destination arbitration according to known techniques.

[0021] Host bridge **16** includes dedicated interrupt (e.g., IRQ) ports **38** through which interrupt request signals (e.g., IRQ signals) are received from interrupt request lines **40**. Interrupt ports **38** may be pins, other structure, or simply conductors. Interrupt controller **34** receives the interrupt request signals from ports **38**. In one embodiment, interrupt ports **38** are considered part of interrupt controller **34** (and, therefore, also part of host bridge **16**). In another embodiment, interrupt ports **38** are considered part of host bridge **16**, but not interrupt controller **34**. The difference is not important so long as interrupt controller **38** receives interrupt request signals.

[0022] Host bridge **16** also includes address, data & other ports **42** through which address, data, and other signals are received from or provided to address, data & other lines **44**. Ports **42** may be pins, other structure, or any other conductor. Ports **38** and **42** may be simply continuations of lines **40** and **44**. Interrupt controller **34** receives at least some of the address, data, and/or other signals passing through ports **42**. Some or all of the address, data, and other signals received at ports **42** are used in host bridge **16** and elsewhere for various purposes other than interrupts. Lines **40** and **44**, as well as the various other lines described herein, may be conventional conductor traces or various other forms of conductors. Depending on the embodiment of the invention, lines **40** and **44** maybe considered part of or separate from bus **20**.

[0023] System **10** includes peripheral devices that may interrupt processor **12** through providing interrupt request signals to interrupt controller **34**. Examples of peripheral devices and related interconnections are illustrated in **FIG. 1**. Peripheral device **50** is coupled to I/O bus **20** through interrupt line(s) **62** and through address, data & other lines **64**. To interrupt processor **12**, peripheral device **50** provides an interrupt request signal (e.g., INTA#) on interrupt line(s) **62**. The interrupt request signal is passed through bus **20** and interrupt lines **40** to interrupt ports **38**. Interrupt controller **34** responds to the interrupt request signal by providing an appropriate interrupt message to processor **12** or another processor (not shown in **FIG. 1**).

[0024] Peripheral device **52** is coupled to I/O bus **20** through address, data & other lines **68**, but not through interrupt line(s). To interrupt processor **12**, peripheral device **52** provides an interrupt request signal on address, data & other lines **68**. In one embodiment of the invention, discussed in greater detail below, the interrupt request signal involves a PCI write cycle. The interrupt request signal is passed through bus **20** and address, data & other lines **44**. Interrupt controller **34** responds to the interrupt request signal by providing an appropriate interrupt message to processor **12** or another processor (in the case of a multi-processor system not shown in **FIG. 1**).

[0025] Accordingly, host bridge **16** may provide interrupt messages to processor **12** or another processor in response to interrupt request signals from two types of peripheral devices. A first type of peripheral device (e.g., peripheral device **50**) provides interrupt request signals (e.g., INTA#) through dedicated interrupt line(s). The interrupt request signals are received by interrupt controller **34** through interrupt ports **38**. A second type of peripheral device (e.g., peripheral device **52**) provides interrupt request signals (e.g., including a PCI write cycle) through, for example, address and data lines. The interrupt request signals are received by interrupt controller **34** through address, data & other lines **44**.

[0026] Peripheral devices **54**, **56**, and **58** illustrate other possible interfaces between peripheral devices and bus **20**. Peripheral device **54** is coupled to bus **20** through an adapter **72**. Adapter **72** may conduct interrupt signals through line(s) **74** and address, data & other signals through lines **76**. Interrupt request signals that are provided on conductors **74** are passed by bus **20** to interrupt lines **40**. Peripheral device **54** is like peripheral device **52** in that it provides interrupt request signals through address, data and other signals, not through an interrupt line(s) **74**. Therefore, in the case of peripheral device **54**, there are no interrupt request signals on interrupt lines **74**. However, a peripheral device like peripheral device **50** could be connected to adapter **72**. In that case, adapter **74** would include interrupt signals on line(s) **74**. Alternatively, some adapters could include only lines **76** and not line(s) **74**. Peripheral devices **56** and **58** are coupled to bus **20** through a bridge **82**. Interrupt request signals are conducted through lines **94**, **90**, and **84**. Address, data & other signals are conducted through lines **98**, **96**, **92**, and **86**.

[0027] **FIG. 2** shows details of one embodiment of host bridge **16**. Address, data & other ports **42** includes address ports **104**, data ports **106**, and other ports **108**. Address, data, and other lines **44** include address lines **114**, data lines **116**, and other lines **118**, which conduct address signals, data signals, and other signals (e.g., enable signals), respectively.

[0028] An interrupt request signal on interrupt lines **40** is provided through ports **38** and conductors **120** to I/O redirection table **128** or other processing circuitry. In response thereto, interrupt controller **34**, including I/O redirection table **128**, provides an interrupt message to a processor. The interrupt message may be provided through serial bus **28** through serial bus controller **144** or through processor bus **26** through encode/decode logic **148**. In the case of sending the interrupt message over processor bus **26**, processor **12** would include decode circuitry to detect the interrupt message and interrupt controller **24** would understand the message.

[0029] In response to receiving an interrupt request signal, at least a portion of which is in the form of address signals, interrupt controller **34** provides an interrupt message to serial bus **28** or processor bus **26**. In one embodiment, host bridge **16** can direct the interrupt message either through serial bus **28** or processor bus **26** depending on a bit in control logic **130**.

[0030] The interrupt message over processor bus **26** could include an address identifying the processor to receive the interrupt. Host bridge **16** could include lowest priority redirection circuitry to redirect the interrupt to the processor having the lowest priority in the case of a multi-processor system. The circuitry could keep track of task priorities of

the various processors in a multiprocessor system. Interrupt controller **34** or other circuitry in host bridge **16** could detect whether processor **12** includes serial bus capabilities and/or the ability to accept interrupt messages by processor bus **26**. In the case where processor **12** does not include an interrupt controller and decode circuitry that understands an interrupt message over processor bus **26**, interrupt controller **34** could direct the interrupt message over serial bus **28** rather than over processor bus **26**. Host bridge **16** may include queues (not shown) to hold various interrupt signals and other signals. Interrupt controller **34** may include queues to hold interrupt request signals. Control logic **130** assists in various functions of interrupt controller **34**.

[0031] An interrupt request signal may be provided in the form of address and data signals (and perhaps other signals) through ports **42** and captured by interrupt controller **34**. In such a case, decoding logic **122** decodes all or part of the address and data signal bits as being an interrupt request signal. In one embodiment, decoding logic **122** provides a decoded signal on conductors **124**. In one embodiment, the decode signal may be an assertion or a deassertion signal. The interrupt request assertion/deassertion signals on conductors **124** may be the same as the interrupt request signals on conductors **120**. In that case, I/O redirection table **128** could treat the signals identically.

[0032] Referring to **FIG. 3**, in one embodiment, decoding logic **122** includes an address decoder **158** and a data decoder **160**. If a particular address or an address within a particular range is received, address decoder **158** provides a signal to control logic **130** on conductor(s) **162** indicating that an interrupt request signal is being provided to interrupt controller **34** through address and data lines **152** and **154**, which are connected to lines **114** and **116**. In one embodiment, an address indicating an interrupt request signal includes a base plus an offset. As an example, the base could (where h=hex). The offset could be *20h*. The base may be programmable by the processor, operating system, or other hardware or software. Control logic **130** provides an enabling signal on conductor(s) **164** to data decoder **160**. In one embodiment, data decoder **160** decodes the 8 least significant bits (LSBs) of the data signal and asserts one of X decode output lines **124**, depending on the state of the data bits. If there are 8 data bits, there may be up to 256 decode output lines **124**.

[0033] Holding registers **170** include a register for each one of decode output lines **124**. Each of the holding register holds the voltage state on a corresponding one of decode output lines **124**. In turn, lines **172** provide signals representing the voltage state held in holding registers **170**. A holding register is set (e.g., has a logic high voltage) through an assertion signal on the corresponding one of lines **124** and is reset through a deassertion signal on the corresponding one of lines **124**. The difference between the assertion and deassertion signals may be merely opposite polarity. In one embodiment, a different address on conductors **152** controls whether an assertion or deassertion signal is provided on decode output lines **124**. In another embodiment, different data signals on conductors **154** control whether an assertion or deassertion signal is provided on a particular one of decode output lines **124**.

[0034] Referring to **FIGS. 3 and 4**, lines **172** include lines **172-0, 172-1, . . . , 172-X-1**, each connected to a different one of holding registers **170**. Interrupt lines **120** include interrupt lines **120-0, 120-1, 120-2, . . . , 120-N-1**, each connected to a different one of interrupt ports **38**. In the embodiment of **FIG. 4**, I/O redirection table **128** includes X entries, which each include a "send pending" (SP) bit (which may be called a delivery status bit). When an SP bit is set, an interrupt message is sent to a processor. SP bits **0-N-1** are set (e.g., to a logic high voltage) when the output of a corresponding OR-gate **190, 192, 194, . . . , 196** is asserted. The OR-gates have inputs of one of lines **120** and one of lines **172**. Accordingly, an interrupt signal to either one of ports **38** or to decoding logic **122** may cause one of SP bits **0-N-1** to be set. For example, SP bit **0** is set when either interrupt line **120-0** or line **172-0** is set. (The OR-gates could be replaced with other logic if SP bits are set through a low voltage. There could be inverters between interrupt ports **38** and the OR gates.) SP bits **N-X-1** are set when a corresponding one of lines **172-N-172-X-1** is asserted. In this way, there may be a greater number of SP bits than interrupt ports **38**. (Note that in the some embodiments and in certain circumstances, the states of the SP bits **0-X-1** may be controlled by signals other than those from lines **120** or **172**.)

[0035] Interrupt controller **34** may support scalability for edge triggered interrupt request signals. In the case of edge triggered interrupts on lines **152** and **154**, data decoder **160** assserts one of lines **124**. The corresponding one of holding registers **170** is set, causing a corresponding one of lines **172** to be asserted. Assertion of one of lines **172** causes the corresponding one of SP bits is set. When the SP bit is set, the particular one of holding registers **170** is reset through conductors **178**. This I/O redirection entry may be then entered into the interrupt delivery rotation scheme to be delivered at the appropriate time. There is no need to initiate an interrupt request deassertion register operation when the interrupt event is removed, because the activation of the signal itself may indicate that one and only one interrupt event will be signaled. As with the input pin scheme, the SP bit of an interrupt defined as edge triggered may be reset when the interrupt has been successfully delivered on the associated message mechanism. If multiple interrupt request assertion register operations are received to the same I/O redirection table entry before the interrupt has been delivered to the destination only one interrupt event may be detected. This behavior is consistent with the dedicated pin scheme.

[0036] With respect to level triggered interrupts, when a device signals an interrupt for a line that is shared by multiple devices, that device may issue an interrupt request operation on the first activation of the interrupt. When the interrupt signal goes inactive, the device may issue an interrupt request deassertion message to interrupt controller **34**. Interrupt controller **34** maintains the activation of the corresponding holding register bit until the deassertion message is received. The constraint of this mechanism is that both the device collecting the input events and the interrupt controller are cognizant that the interrupt request is configured as a level triggered interrupt event. For these events, the interrupt request deassertion register transactions may be required for correct operation. Signals on lines **116** or **118** may indicated whether an edge or level triggered interrupt signal is involved.

[0037] In the embodiment of **FIG. 4**, I/O redirection table **128** also includes interrupt request register (IRR) bits **0, 1,**

. . . , X-1, which are used in the case of level triggered interrupts. The SP bit is reset when the IRR bit is set. The IRR bit is set when an interrupt message is accepted by the processor. Either a level assert message is issued and not retried on processor bus **16** or a message on serial bus **28** is accepted. The IRR bit is reset when an EOI message is received. For both serial and parallel bus delivery, the IRR bit is reset with a write to the corresponding EOI register, the vector of which matches the vector field of the redirection entry.

[0038] When an interrupt is serviced, a deassertion signal is provided by the peripheral device to decode logic **122**. If after the IRR bit is reset, the corresponding holding register is set, then there is another interrupt waiting to be acknowledged. The corresponding SP bit is then set.

[0039] **FIG. 5** illustrates details of one embodiment of peripheral device **52**. Address, data & other lines **68** include address lines **180**, data lines **182**, and other lines **184**. An interrupt controller **17415** provides interrupt request signals to at least some of the bits of address lines **180**. The interrupt request signal may also include bits on data lines **182** and/or other lines **184**. In one embodiment, interrupt controller **174** includes a data register(s) the contents of which control whether peripheral device **52** sends interrupt request signals in the form of an interrupt signal to a dedicated interrupt port or in the form of address and data signals, and particular details regarding the signals.

[0040] An advantage of the invention is that level triggered interrupts on interrupt lines **40** may be replaced by write cycle messages or other address signal based messages. In one embodiment, the write cycle message may identify the origin of the interrupt request. Further, the number of send pending bits may be easily increased without the adding dedicated interrupt lines.

[0041] Interrupt controller **34** may support multiple interrupt request signal input mechanisms. However, in order to avoid any race conditions that may occur, in one embodiment, only one mechanism per interrupt request signal is supported at a given time. The interaction of the various arrival times and rates may be identical to the dedicated port (e.g., pin) approach. Multiple activations of an event from a device will elicit the interrupt request assertion/deassertion signal which may provide a model consistent with the operation of the dedicated port.

[0042] Each interrupt controller may have a unique address for configurability and any access to this address space, regardless of the initiating resource may reach the final destination. As an example, if a system contains two I/O buses, the first contains the interrupting device and the second contains the interrupting controller. The interrupting device, through the unique address of the interrupting controller, may be capable of directing an interrupt request assertion signal to the interrupting controller. Note that this messaging scheme does not require a 'sidecar' path for interrupts that is different than the path to main memory. Signaling the interrupt request assertion signal may have the effect of flushing any previous write transactions.

[0043] Additional Information and Embodiments

[0044] The specification does not describe or illustrate various well known components, features, and conductors, a discussion of which is not necessary to understand the invention and inclusion of which would tend to obscure the invention. Furthermore, in constructing an embodiment of the invention, there are design tradeoffs and choices, which would vary depending on the embodiment. There are a variety of ways of implementing the illustrated and unillustrated components.

[0045] The borders of the boxes in the figures are for illustrative purposes and do not restrict the boundaries of the components, which may overlap. The relative size of the illustrative components does not to suggest actual relative sizes. Arrows show principle data flow in one embodiment, but not every signal, such as requests for data flow. As used herein "logic" does not mean that software control cannot be involved. The term "conductor" is intended to be interpreted broadly and includes devices that conduct although they also have some insulating properties. There may be intermediate components or conductors between the illustrated components and conductors.

[0046] The interrupt message provided by interrupt controller **34** to interrupt controller **24** may be somewhat altered in host bridge **16**, processor bus **26**, and/or serial bus **28** prior to it being received by interrupt controller **24**. For example, bits of the interrupt message provided by interrupt controller **34** could be inverted or encoded. Address bits could be added by encode/decode logic or other circuitry.

[0047] In one embodiment, host bridge **16** does not include the capability to send interrupt messages over processor bus **26**. In that embodiment, conductors might not connect I/O redirection table **128** to encode/decode logic **148**. As shown in **FIG. 5**, in another embodiment, host bridge **16** does not include the capability to send interrupt messages over serial bus **28**. In that embodiment, serial bus controller **144** and associated conductors are not included in host bridge **16**.

[0048] In one embodiment, a signal on processor bus **26** is two phase signal. In the first phase, if an Aa3# bit is 0, the interrupt transaction type is fixed (directed); if the Aa3# bit is 1, the type is redirected or EOI. In the second phase, Ab5# and Ab6# bits of 00 indicate physical destination mode, and Ab5# and Ab6# bits of 01 indicate logical destination mode. Ab5# and Ab6# bits of 11 indicate an EOI. Aa3# and Ab6# bits of 0 and 1 and Aa3#, Ab5#, and Ab6# bits of 110 are reserved.

[0049] The holding registers and SP bits may be in parallel with respect to conductors **124**.

[0050] Interrupt controller **34** does not have to be part of host bridge **16**. There may be an interrupt router between the peripheral devices (interrupting agents or PCI devices) and the interrupt controller. Decode logic **122** may be outside interrupt controller **34**.

[0051] The phrase "in one embodiment" means that the particular feature, structure, or characteristic following the phrase is included in at least one embodiment of the invention, and may be included in more than one embodiment of the invention. Also, the appearances of the phrase "in one embodiment" in various places in the specification are not necessarily all referring to the same one embodiment.

[0052] The term "connected" and "coupled" and related terms are used in an operational sense and are not necessarily limited to a direct connection or coupling. If the

specification states a component or feature "may", "can", "could", or "might" be included or have a characteristic, that particular component or feature is not required to be included or have the characteristic. The term "responsive" includes completely or partially responsive.

[0053] Those skilled in the art having the benefit of this disclosure will appreciate that many other variations from the foregoing description and drawings may be made within the scope of the present invention. Accordingly, it is the following claims including any amendments thereto that define the scope of the invention.

What is claimed is:

1. An apparatus, comprising:

address and data ports to receive an interrupt request signal in the form of address signals and data signals;

decode logic to receive at least some of the address signals and data signals and to provide a decoded signal at one of several decode output lines of the decode logic; and

a redirection table including a send pending bit that is set responsive to the decode signal.

2. The apparatus of claim 1, further including a holding register that are set in response to assertion of the decoding signal.

3. The apparatus of claim 1, wherein the interrupt request signal is in the form of the address signals, the data signals, and other signals.

4. The apparatus of claim 1, wherein the apparatus is a bridge.

5. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal;

address and data ports capable of receiving an interrupt request signal in the form of address signals and data signals;

decode logic to provide a decode signal at one of several decode output lines in response to reception of the interrupt request signal in the form of address signals and data signals;

a redirection table including a send pending bit to be set in response to either the interrupt request signal at the dedicated interrupt ports or in response to the decode signal.

6. The apparatus of claim 5, wherein OR gates are positioned between the interrupt ports, some of the decode lines, and the redirection table.

7. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal from interrupt request lines;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

redirection and control circuitry coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

8. The apparatus of claim 7, wherein the interrupt request signal provided by the decode logic is identical to the interrupt request signal provided by the dedicated interrupt ports.

9. The apparatus of claim 7, wherein the redirecton and control circuitry includes an I/O redirection table.

10. The apparatus of claim 7, wherein the decode logic is included in an interrupt controller.

11. An apparatus, comprising:

dedicated interrupt ports to receive an interrupt request signal from interrupt request lines;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

an I/O redirection table coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

12. The apparatus of claim 11, wherein the interrupt request signal provided by the decode logic is identical to the interrupt request signal provided by the dedicated interrupt ports.

13. The apparatus of claim 11, wherein the decode logic is included in an interrupt controller.

14. A computer system, comprising:

a processor;

an I/O bus;

peripheral devices connected to the I/O bus; and

a bridge including:

dedicated interrupt ports to receive an interrupt request signal;

address and data ports to receive address and data signals;

decode logic to receive at least some of the address and data signals and to decode an interrupt request signal from them; and

redirection and control circuitry coupled to the dedicated interrupt request ports and the decode logic to receive the interrupt request signal from the interrupt ports and the interrupt request signal from the decode logic and in response thereto to provide an interrupt message.

15. The system of claim 14, wherein at least one of the peripheral devices provide interrupt request signals through the dedicated interrupt ports and at least one of the peripheral devices provides interrupt request signals in the form of write cycles through the address and data ports.

16. The system of claim 14, wherein at least one of the peripheral devices can provide address and data signals to the decode logic for decoding an interrupt request signal from them.

* * * * *